

AD-A277 861



DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

This document is estimated to average 1 hour per reviewer, including the time for reviewing instructions, searching existing data sources, gathering and reviewing the collection of information, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including this burden estimate, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Avenue, Washington, DC 20540, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

2. REPORT DATE 1 March 1994		3. REPORT TYPE AND DATES COVERED Final Technical Report 11/1/90-11/30/94	
4. TITLE AND SUBTITLE Adaptive Methods for Compressible Flow		5. FUNDING NUMBERS AFOSR-91-0063 2304/CS 61102F	
6. AUTHOR(S) Marsha Berger		94-10516 099950 AEOSR-TR- 94 0131	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) New York University Courant Institute of Mathematical Sciences 251 Mercer Street New York, NY 10012			
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM 110 Duncan Avenue, Suite B115 Bolling AFB, DC 20332-0001		10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFOSR-91-0063	
11. SUPPLEMENTARY NOTES		DTIC ELECTE APR 07 1994 S D	
12a. DISTRIBUTION / AVAILABILITY STATEMENT Unlimited Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE F	
13. ABSTRACT (Maximum 200 words) The goal of this research is the development of adaptive computational methods to numerically simulate fluid flows around complex configurations in an automatic fashion. Grid generation continues to be a huge impediment for computer simulations of realistic fluid flows. This is true for both body-fitted structured grid solvers and unstructured grid approaches. We are developing a Cartesian grid representation of the geometry, where the object is simply "cut" out of the Cartesian grid. We are also investigating the suitability of adaptive methods on parallel computers. DTIC QUALITY INSPECTED 3			
14. SUBJECT TERMS Adaptive mesh refinement, compressible fluid flows, Cartesian meshes.		15. NUMBER OF PAGES 8	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

94 4 6 042

AFOSR-TR- 94 0131

Adaptive Methods for Compressible Flow

Final Report AFOSR-91-0063

Marsha J. Berger
Courant Institute
251 Mercer Street
New York, NY 10012

Accession For	
NTIS	CRA&I
DTIC	TAB
Unannounced	
Justification	
By	
Distribution	
Availability	
Dist	Special
A-1	

1 Abstract

The goal of this research is the development of adaptive computational methods to numerically simulate fluid flows around complex configurations in an automatic fashion. Grid generation continues to be a huge impediment for computer simulations of realistic fluid flows. This is true for both body-fitted structured grid solvers and unstructured grid approaches. We are developing a Cartesian grid representation of the geometry, where the object is simply "cut" out of the Cartesian grid. We are also investigating the suitability of adaptive methods on parallel computers.

Here we summarize results of our previous work in the last three years, supported by AFOSR grant 91-0063.

2 Summary of Accomplishments

Our research in the last three years has concentrated on three fronts: (i) we have continued to develop the basic adaptive methodology for compressible flows in the context of hyperbolic conservation laws, (ii) we have developed a strategy for simulating time-dependent flows around complex geometries using a Cartesian grid representation in two space dimensions, and (iii) we have studied the suitability of local adaptive mesh refinement for massively parallel architectures such as the Connection Machine. I will describe these projects in turn.

The development of a (serial) three dimensional adaptive mesh refinement (henceforth AMR) algorithm for hyperbolic conservation laws has been completed. This work is in

collaboration with John Bell and Michael Welcome, at Livermore National Laboratory, and Jeff Saltzman, of Los Alamos National Laboratory. To test the algorithm we conducted numerical experiments of the interaction of a shock and an ellipsoidal bubble of Freon (see figure 1. This is analogous to the laboratory experiments of Haas and Sturtevant. Our tests indicate that AMR reduced the computational cost by a factor of at least 20 over uniform fine grid computations of the same resolution. This work is described in [1]

Although most of the algorithm development was a straightforward extension of the two dimensional case, a new grid generation algorithm was developed. Our previous grid generation algorithm worked remarkably well at creating grids with efficiency ratings of approximately 50%, (i.e. at least half the points in the new fine grid needed to be refined), using no geometric information at all. (The rest of the points are included to make a rectangular fine grid, but didn't need to be refined based on the error estimate criterion). If a fine grid was inefficient, it simply bisected the grid in the long direction, the flagged cells were partitioned into their respective halves, and two new fine grids resulted. Unfortunately, when requesting efficiencies around 80%, this algorithm produced unacceptably many tiny grids. Since memory (to store the refined grids) and CPU time (to integrate them) is at a premium in three dimensions, we developed a better grid generation algorithm. The new procedure can be viewed as a "smart" bisection algorithm. Using techniques from computer vision and pattern recognition, such as a variation of the Marr-Hildreth operator and coordinate-based signatures, our new algorithm can obtain efficiencies around the 80% level, even if the feature is not aligned with a coordinate direction. This was described in [2].

In collaboration with Prof. Randy LeVeque at the University of Washington we have developed an algorithm for computing time dependent flows around complex geometries using a non-body-fitted Cartesian grid. This strategy fits in naturally with our previously developed AMR strategy of using locally uniform meshes. We retain the advantages (efficiency and accuracy) of uniform grids and are able to resolve fine scale flow features induced by complex geometries. We use our previously developed adaptive mesh refinement algorithm to achieve accuracy comparable to the body-fitted meshes, where grid points can be bunched in an a priori manner to improve the accuracy of the solution.

We have developed a rotated difference scheme for use at the irregular boundary cells. Essentially, this difference scheme uses an auxiliary coordinate system that is locally normal and tangential to the boundary. This "artificial" grid is obtained at each interface by creating a box extending a distance h away from the interface in the normal and tangential directions. Solution values for the new box are obtained by conservatively averaging the states in the underlying Cartesian grid. By differencing over a box of size h rather than the irregular neighboring cell with (potentially orders of magnitude) less cell volume, we retain stability using a time step Δt based on the uniform grid cells. Figure 2 shows a snapshot of a time-dependent computation of shocked flow around two cylinders. Notice that the

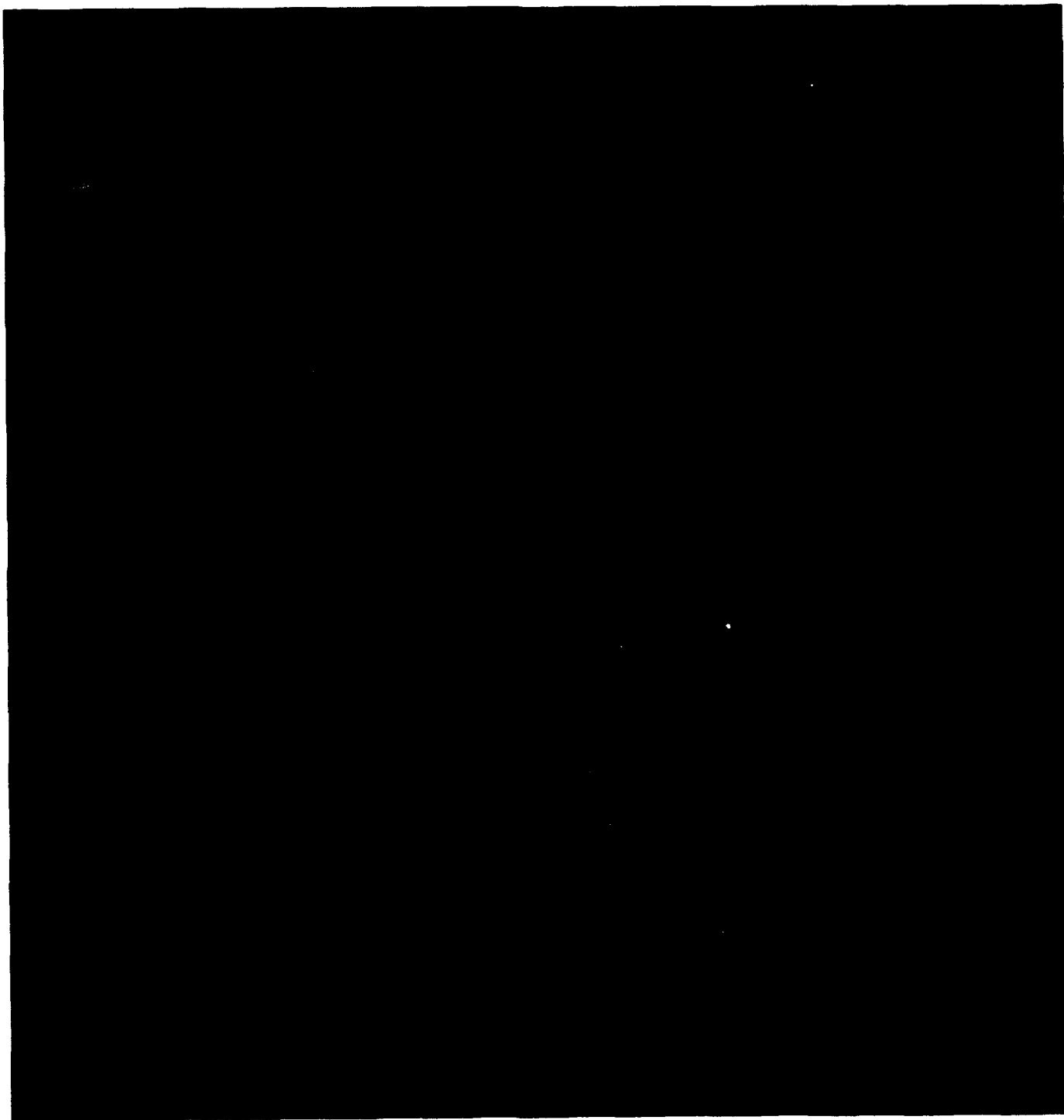


Figure 1: Interaction of a shock and ellipsoidal bubble.

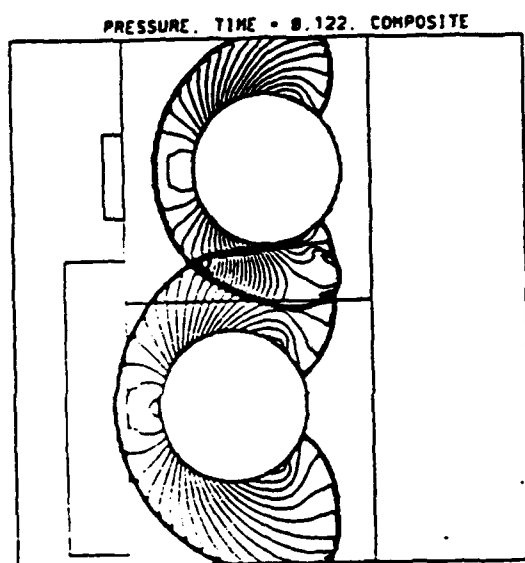
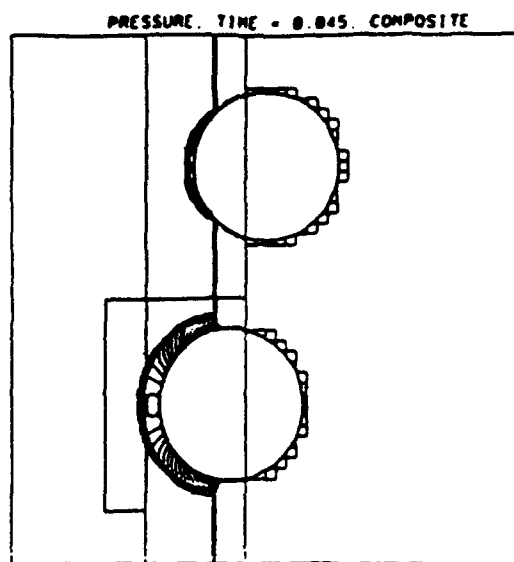
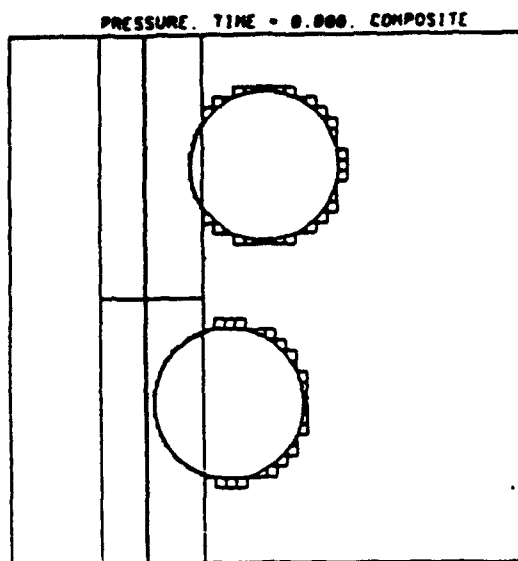
pressure plots around the cylinders show smooth pressure despite the irregularity of the cut cells adjacent to the cylinders.

In more recent work with John Melton (at NASA Ames Research Center), we are developing a steady state compressible flow solver around objects with complex geometry using a Cartesian grid. With this work, since there is no time stepping stability limitation, we are concentrating on the accuracy question of irregular grid cells, and focusing on the issue of automating the geometry. We have developed an approach, using a given surface triangulation, to generate the volume information and data structures needed in the Cartesian grid program [4]. In addition, we have the beginnings of an automatic interface using a CAD/CAM system. Figure 3 shows a sample calculation that illustrates the Cartesian geometry.

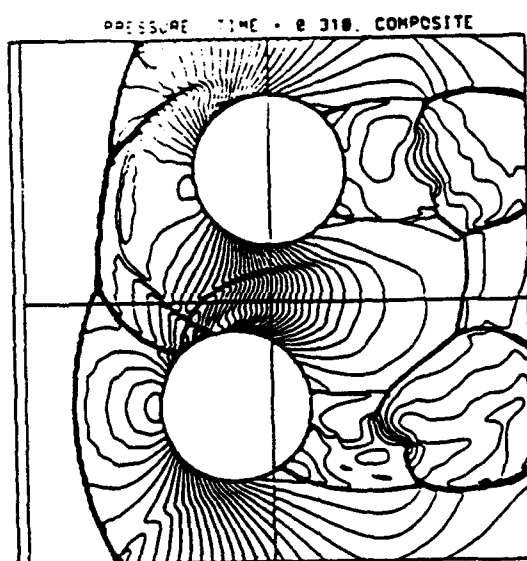
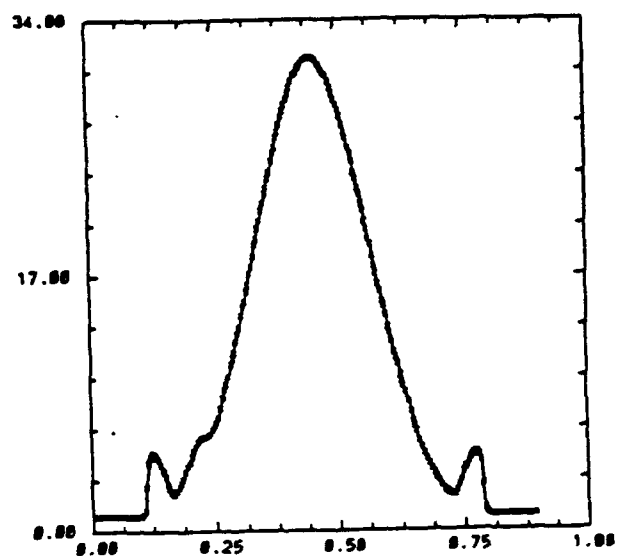
As a preliminary part of this study, we have compared the use of hierarchical meshes with rectangular indexing versus a completely unstructured linked list implementation. An unstructured grid data structure has a lot of overhead associated with it, since each tetrahedron points to all its neighbors, its faces, its edges, etc. Typical numbers are approximately 80 to 120 words of storage per grid point. A grid based data structure has much less overhead, but needs to refine more cells than the absolute minimum to form the rectangles. We have run some experiments comparing the approaches in three dimensions, for flow around a wing, and a full aircraft. Although this is very simple it appears to have not been done before. For the wing test case, our results show that although the regular approach refines one and a half times as many points, it has a factor of 5 less overhead, so it is still preferable. For the full aircraft, approximately twice as many points are used in the flow solver, but again with less overhead. Additionally, the regular grid scheme vectorizes without using indirect address and scatter gathers, so performance should be the same if not slightly better than the unstructured approach as well.

Together with Jeff Saltzman from Los Alamos National Laboratory, we have developed a fully adaptive two dimensional local mesh refinement algorithm for the Euler equations on the CM-2 and Cm-5. To our knowledge, this is the first time such an adaptive method has been implemented for structured meshes on a massively parallel machine. Again we use the AMR approach to adaptive mesh refinement; a collection of logically rectangular meshes makes up the coarse grid, refinements cover a subset of the domain and use smaller rectangular grid patches. (There is no complex geometry in this implementation). Many of the original design choices in AMR were based on considerations of vectorization. The question was whether this approach was still feasible, and even advantageous on a data parallel massively parallel architecture.

The main issues in adapting the algorithm for a data parallel environment were partitioning the data to fill the machine (load balancing, rather simple on rectangular grids), and minimizing communication (which is the main issue for our hierarchical data structure). We have developed a data layout scheme which preserves locality between fine and coarse grids



BODYPLOT. TIME = 0.122. COMPOSITE 1



BODYPLOT. TIME = 0.310. COMPOSITE 2

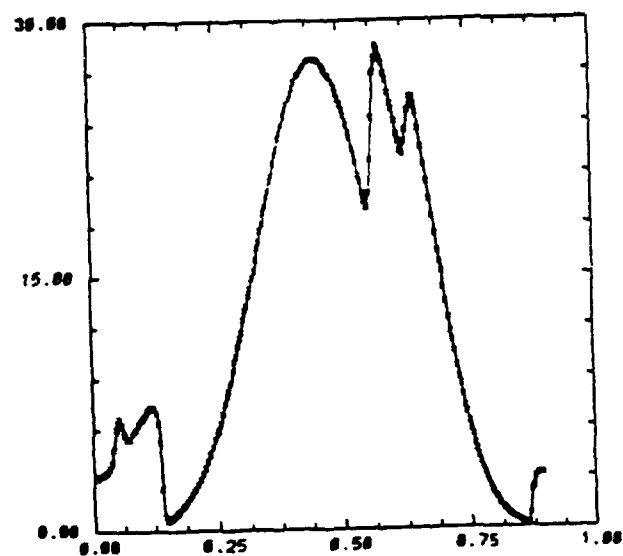


Figure 2 shows pressure plots of the flowfield and the pressure along the cylinders, at several different times. The location of embedded fine grids is indicated on the contour plots.

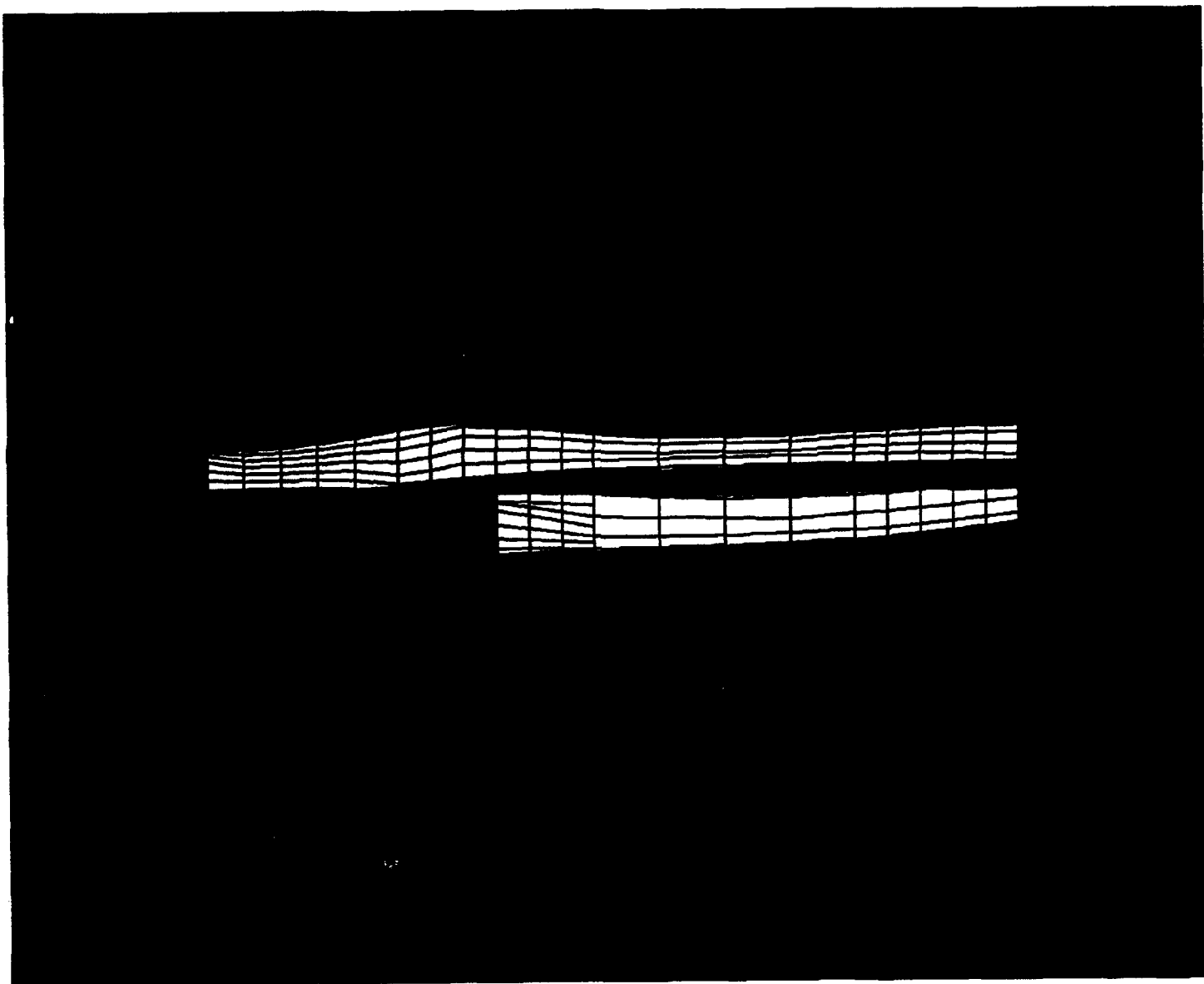


Figure 3: Cartesian mesh representation of an airplane.

and minimizes global (router) communication. This data layout turned out to be an important component in equidistributing the communication (rather than the more typical load balancing of computation) even on the CM-5, with its unusual fat-tree interconnection. The two-dimensional CM-2 results are described in [3]. Essentially, we obtained performance on an 8K CM-2 that was equivalent to one head of a Cray Y-MP. The three-dimensional CM-5 research is still underway. Eventually, we will implement a parallel version of adaptive mesh refinement with complex geometry. This will not be simply a straightforward implementation, since the workloads have much more non-uniform, depending on whether a cell is adjacent to a solid object, or at the boundary of a finer or coarser level grid.

In addition to the above research, the research by Anders Szepessy on adaptive finite element methods for compressible fluid flow continued. For adaptive flow calculations, one needs:

1. a robust mesh generator
2. a stable and reasonable accurate discretization method,
3. an adaptive refinement criteria.

The work has concentrated on the less well understood areas (1) and (3). In joint work with Jonathan Goodman and Margaret Symington, we have been developing a new mesh generator based on successive not necessarily isotropic refinements using high aspect ratio elements around shocks and boundary layers. The most commonly used method for generating anisotropic meshes is the advancing front technique, which is not very robust in my experience. Computations with our new program show promise. The mesh generator is very robust.

Currently we are working on techniques to improve the convergence rate when solving the discrete equations. We have been improving the data structure in the program to more efficiently handle multilevel techniques. The hierarchical multilevel structure naturally obtained in the successive refinements is known to work well in the case of isotropic refinements. In our case of anisotropic discretizations and high aspect ratio elements the condition numbers are even worse as compared to the isotropic case. Preliminary tests using multilevel techniques indicate that one can obtain convergence rates independent of the mesh size also with the high aspect ratio elements in our program.

Reacting shock waves, in contrast to non-reacting shocks, sometimes need high resolution to capture the correct physical behavior. It is therefore of interest to apply our program to reactions. In previous work with Claes Johnson we have constructed adaptive algorithms based on a posteriori error estimates for non-reacting shocks using mainly isotropic meshes. We are currently studying generalizations to reactions, relaxation effects and anisotropic meshes including implementation aspects.

References

- [1] J. Bell, M. J. Berger, J. Saltzman, and M. Welcome. Three dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Stat.*, 15:127-138, January 1994.
- [2] M. J. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Trans. Sys. Man and Cyber.*, 21:1278-1286, Sept./Oct. 1991.
- [3] M. J. Berger and J. Saltzman. AMR on the CM-2. In *Proceedings of Workshop on Adaptive Computational Methods*, Troy, NY, June 1992. To appear, *Applied Num. Math.*
- [4] J. Melton, F. Enomoto, and M. Berger. 3d automatic Cartesian grid generation for euler flows. *AIAA-93-3386*, July 1993.

Unstructured Multigrid through Agglomeration

V. Venkatakrishnan

Computer Sciences Corporation

M.S. T045-1, NAS Applied Research Branch

NASA Ames Research Center, Moffett Field, CA 94035

D. J. Mavriplis

ICASE

M.S. 132C

NASA Langley Research Center, Hampton, VA 23681

M.J. Berger

Courant Institute of Mathematical Sciences

New York University, New York, NY 10012

Abstract

In this work the compressible Euler equations are solved using finite volume techniques on unstructured grids. The spatial discretization employs a central difference approximation augmented by dissipative terms. Temporal discretization is done using a multistage Runge-Kutta scheme. A multigrid technique is used to accelerate convergence to steady state. The coarse grids are derived directly from the given fine grid through agglomeration of the control volumes. This agglomeration is accomplished by using a greedy-type algorithm and is done in such a way that the load, which is proportional to the number of edges, goes down by nearly a factor of 4 when moving from a fine to a coarse grid. The agglomeration algorithm has been implemented and the grids have been tested in a multigrid code. An area-weighted restriction is applied when moving from fine to coarse grids while a trivial injection is used for prolongation. Across a range of geometries and flows, it is shown that the agglomeration multigrid scheme compares very favorably with an unstructured multigrid algorithm that makes use of independent coarse meshes, both in terms of convergence and elapsed times.

1 Introduction

Multigrid techniques have been successfully used in computational aerodynamics for over a decade [1,2]. The main advantage of the multigrid method when solving steady flows is the enhanced convergence while requiring little additional storage. In addition, multigrid can be used in conjunction with any convergent base scheme, with adequate care exercised in constructing proper restriction and prolongation operators between the grids. Perhaps the biggest advantage of multigrid is the fact that it deals directly with the nonlinear problem without requiring an elaborate linearization and the attendant storage required to store the matrix that arises from the linearization. Thus, multigrid techniques have enabled the practical solution of complex aerodynamic flows using millions of grid points.

The initial efforts in multigrid were directed towards the solution of flows on structured grids where coarse grids can easily be derived from a given fine grid. Typically, this is done

by omitting alternate grid lines in each dimension. These ideas have been extended to triangular grids in two dimensions and to tetrahedral meshes in three dimensions [3,4,5,6]. In previous work by the second author, a sequence of unnested triangular grids of varying coarseness is constructed [3]. Piecewise linear interpolation operators are derived during a preprocessing step by using efficient search procedures. The residuals are restricted to coarse grids in a conservative manner. It has been shown that such a scheme can consistently obtain convergence rates comparable to those obtained with existing structured grid multigrid methods. For complex geometries, especially in three dimensions, however, constructing coarse grids that faithfully represent the complex geometries can become a difficult proposition. Thus, it is often desirable to derive the coarse grids directly from a given fine grid.

The agglomeration multigrid strategy has been investigated by Lallemand et al. [7] and Smith [8]. Lallemand et al. use a base scheme where the variables are stored at the vertices of the triangular mesh, whereas Smith uses a scheme that stores the variables at the centers of triangles. In the present work, a vertex-based scheme is employed. Two dimensional triangular grids contain twice as many cells as vertices (neglecting boundary effects), and three dimensional tetrahedral meshes contain 5 to 6 times more cells than vertices. Thus, on a given grid, a vertex scheme incurs substantially less computational overhead than a cell-based scheme. Increased accuracy can be expected from a cell-based scheme, since this involves the solution of a larger number of unknowns. However, the increase in accuracy does not appear to justify the additional computational overheads, particularly in three dimensions.

The main idea behind the agglomeration strategy of Lallemand et al. [7] is to agglomerate the control volumes for the vertices using heuristics. The centroidal dual, composed of segments of the median of the triangulation, is a collection of the control volumes over which the Euler equations in integral form are solved. On simple geometries, Lallemand et al. were able to show that the agglomerated multigrid technique performed as well as the multigrid technique which makes use of unnested coarse grids. However, the convergence rates, especially for the second order accurate version of the scheme, appeared to degrade somewhat. Furthermore, the validation of such a strategy for more complicated geometries and much finer grids, as well as the incorporation of viscous terms for the Navier-Stokes equations remains to be demonstrated. The work of Smith [8] constitutes the basis of a commercially available computational fluid dynamics code, and as such has been applied to a number of complex geometries [9]. However, consistently competitive multigrid convergence rates have yet to be demonstrated.

In the present work, the agglomeration multigrid strategy is explored further. The issues involved in a proper agglomeration and the implications for the choice of the restriction and prolongation operators are addressed. Finally, flows over non-simple two-dimensional geometries are solved with the agglomeration multigrid strategy. This approach is compared with the unstructured multigrid algorithm of Mavriplis [3] which makes use of unnested coarse grids. Convergence rates as well as CPU times on a Cray Y-MP/1 are compared using both methods.

2 Governing Equations and discretization

The Euler equations in integral form for a control volume Ω with boundary $\partial\Omega$ read

$$\frac{d}{dt} \int_{\Omega} \mathbf{u} \, dv + \oint_{\partial\Omega} \mathbf{F}(\mathbf{u}, \mathbf{n}) \, dS = 0. \quad (1)$$

Here \mathbf{u} is the solution vector comprised of the conservative variables: density, the two components of momentum, and total energy. The vector $\mathbf{F}(\mathbf{u}, \mathbf{n})$ represents the inviscid flux vector for a surface with normal vector \mathbf{n} . Equation (1) states that the time rate of change of the variables inside the control volume is the negative of the net flux of the variables through the boundaries of the control volume. This net flux through the control volume boundary is termed the *residual*. In the present scheme the variables are stored at the vertices of a triangular mesh. The control volumes are non-overlapping polygons which surround the vertices of the mesh. They form the *dual* of the mesh, which is composed of segments of medians. Associated with each edge of the original mesh is a (segmented) dual edge. The contour integrals in Equation (1) are replaced by discrete path integrals over the edges of the control volume. Figure 1 shows a triangulation for a four-element airfoil and Figure 2 shows the centroidal dual. Each cell in Figure 2 represents a control volume. The path integrals are computed by using the trapezoidal rule. This can be shown to be equivalent to using a piecewise linear finite-element discretization. For dissipative terms, a blend of Laplacian and biharmonic operators is employed, the Laplacian term acting only in the vicinity of shocks. A multi-stage Runge-Kutta scheme is used to advance the solution in time. In addition, local time stepping, enthalpy damping and residual averaging are used to accelerate convergence. The principle behind the multigrid algorithm is that the errors associated with the high frequencies are annihilated by the carefully chosen smoother (the multi-stage Runge-Kutta scheme) while the errors associated with the low frequencies are annihilated on the coarser grids where these frequencies manifest themselves as high frequencies. In previous work [3], as well as in the present work, only the Laplacian dissipative term (with constant coefficient) is used on the coarse grids. Thus the fine grid solution itself is second order accurate, while the solver is only first order accurate on the coarse grids.

3 Details of agglomeration

The agglomeration (referred to also as coarsening) algorithm is a variation on the one used by Lallemand et al. [7] and is given below:

1. Pick a starting vertex on the surface of one of the airfoils.
2. Agglomerate control volumes associated with its neighboring vertices which are not already agglomerated.
3. Define a front as comprised of the exterior faces of the agglomerated control volumes. Place the exposed edges in a queue.

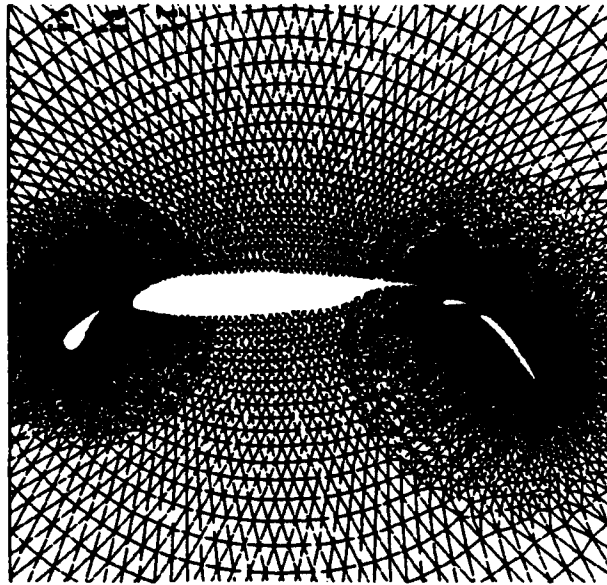


Figure 1: Grid about a four-element airfoil.

4. Pick the new starting vertex as the unprocessed vertex incident to a new starting edge which is chosen from the following choices given by order of priority:
 - An edge on the front that is on the solid wall.
 - An edge on the solid wall.
 - An edge on the front that is on the far field boundary.
 - An edge on the far field boundary.
 - The first edge in the queue.
5. Go to Step 2 until the control volumes for all vertices have been agglomerated.

There are many other ways of choosing the starting vertex in Step 4 of the algorithm, but we have found the above strategy to be the best. The efficiency of the agglomeration technique can be characterized by a histogram of the number of fine grid cells comprising each coarse grid cell. Ideally, each coarse grid cell will be made up of exactly four fine grid cells. The various strategies can be characterized by how close they come to this ideal case. One variation is to pick the starting edge randomly from the edges currently on the front. Figure 3 shows a plot of the number of coarse grid cells as a function of the number of fine grid cells comprising them, with our agglomeration algorithm described above, and with the variation. It is clear that our agglomeration algorithm is superior to the variant. The number of coarse grid cells having exactly one fine cell (singletons) is also much smaller with our algorithm compared to the variant. We have also investigated another variation where the starting vertex in Step 4 is randomly picked from the field and this turns out to be much worse. It is possible to identify the singleton cells and agglomerate them with the neighboring cells, but this has not been done.

The procedure outlined above is applied recursively to create coarser grids. Figure 4 shows an example of the agglomerated coarse grid. The boundaries between the control

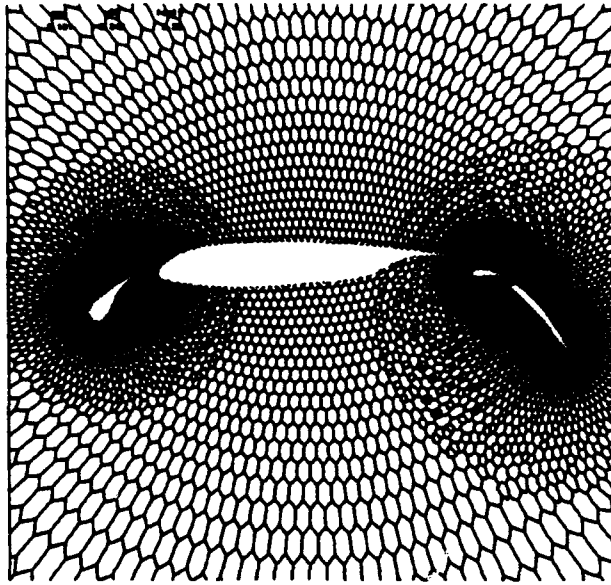


Figure 2: Centroidal dual for the triangulation of Figure 1.

volumes on the coarse grids are composed of the edges of the fine grid control volumes. We have observed that the number of such edges only goes down by a factor of 2 when going from a fine to a coarse grid. Since the computational load is proportional to the number of edges, this is unacceptable in the context of multigrid. However, if we recognize that the multiple edges separating two control volumes can be replaced by a single edge connecting the end points, then the number of edges does go down by a factor of 4. Since only a first order discretization is used on the coarse grids, there is no approximation involved in this step. If a flux function that involved the geometry in a nonlinear fashion were used, such as the Roe's approximate Riemann solver, this is still a very good approximation. It may also be seen from Figure 4 that once this approximation is made, the degree of a node in this graph is still 3 i.e., each node in the interior has precisely three edges emanating from it. Thus the agglomerated grid implies a triangulation of the vertices of a dual graph of the coarse grid. Trying to reconstruct the triangulation is not a good idea, since this may result in a graph with intersecting edges (non planar graph), which leads to non-valid triangulations. If a valid triangulation could always be constructed, it would be possible to use the coarse grid triangulation for constructing piecewise linear operators for prolongation and restriction akin to the non-nested multiple grid scheme [3]. In practice, we have often found the implied coarse grid triangulations to be invalid and therefore the coarse grids are only defined in terms of control volumes. This has some important implications for the multigrid algorithm discussed below.

Since the fine grid control volumes comprising a coarse grid control volume are known, the restriction is similar to that used for structured grids. The residuals are simply summed from the fine grid cells and the variables are interpolated in an area-weighted manner. For the prolongation operator, we use a simple injection (a piecewise constant interpolation). This is an unfortunate but unavoidable consequence of using the agglomeration strategy. A piecewise linear prolongation operator implies a triangulation, the avoiding of which is

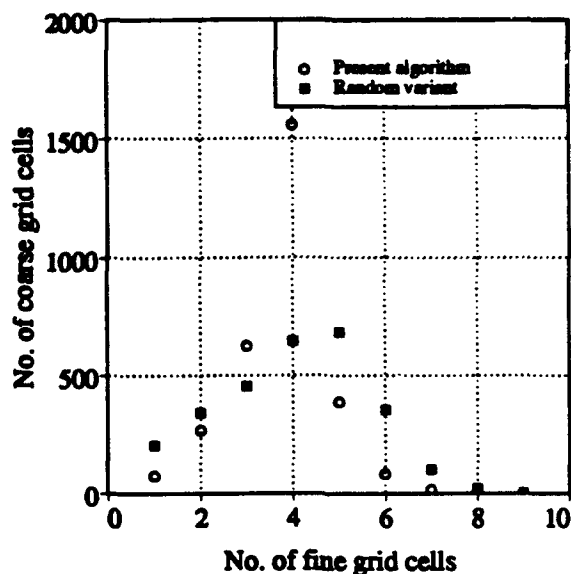


Figure 3: No. of coarse grid cells as a function of the fine grid cells they contain.

the main motivation for the agglomeration. However, additional smoothing steps may be employed to minimize the adverse impact of the injection. This is achieved by applying an averaging procedure to the injected corrections. In an explicit scheme, solution updates are directly proportional to the computed residuals. Thus, by analogy, for the multigrid scheme, corrections may be smoothed by a procedure previously developed for implicit residual smoothing [3]. The implicit equations for the smoothed corrections are solved using two iterations of a Jacobi scheme after the prolongation at each grid level.

The agglomeration step is done as a preprocessing operation on a workstation. It is very efficient and employs hashing to combine the multiple fine grid control volume edges separating two coarse grid cells into one edge. The time taken to derive 5 coarse grids on a Silicon Graphics work station model 4D/25 (20 MHz clock) for the grid shown in Figure 1 with 11340 vertices is 83 seconds.

4 Results and discussion

Results are presented for two inviscid flow calculations and the performance of the agglomerated multigrid algorithm is compared with that of the non-nested multiple grid multigrid algorithm of [3]. The first flow considered is flow over an NACA0012 airfoil at a freestream Mach number of 0.8 and angle of attack of 1.25° . The dual to the fine grid having 4224 vertices is shown in Figure 5. The sequence of unnested grids (not shown) for use with the non-nested multigrid algorithm contains 1088, 288 and 80 vertices, respectively. The agglomerated grids are shown in Figure 6. These grids have 1088, 288 and 80 vertices (regions) as well. Figure 7 shows the convergence histories obtained with the non-nested and agglomeration multigrid algorithms. Both the multigrid strategies employ W-cycles. The convergence histories show that the multigrid algorithm slightly outperforms the agglomeration algorithm. The CPU times required for 100 iterations on the Cray Y-MP/1

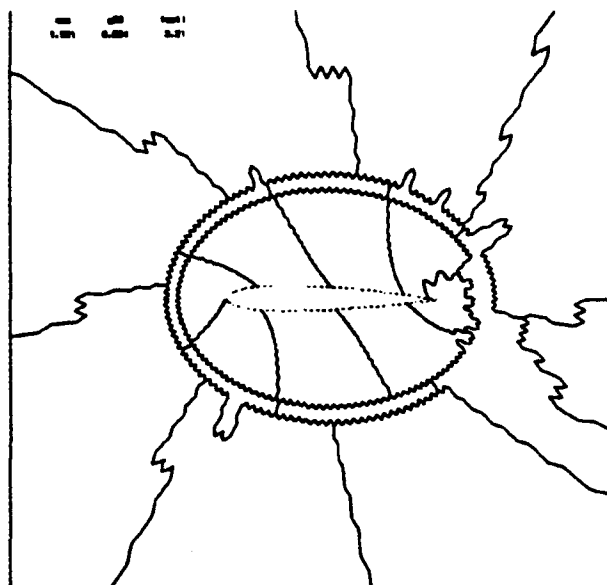


Figure 4: An example of an agglomerated coarse grid.

are 25 and 24 seconds, respectively. Thus the two schemes perform equally well.

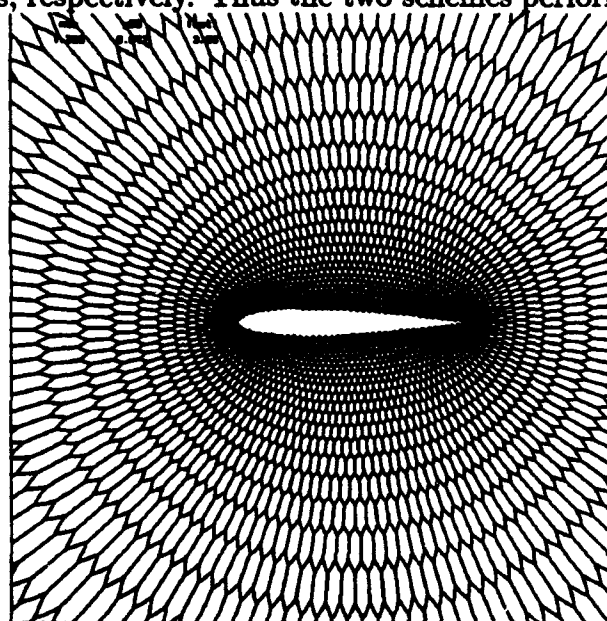


Figure 5: Dual to the fine grid having 4420 vertices.

The next case considered is flow over a four-element airfoil. The freestream Mach number is 0.2 and the angle of attack is 5° . The fine grid has 11340 vertices and is shown in Figure 1. The coarse grids for use with the non-nested multigrid algorithm (not shown) contain 2942 and 727 vertices. The two agglomerated grids are shown in Figure 8. These grids contain 3027 and 822 vertices (regions), respectively. The convergence histories of the non-nested and agglomeration multigrid algorithms are shown in Figure 9. The convergence histories are comparable but the convergence is slightly better with the agglomerated multigrid strategy. This is a bit surprising since the original multigrid algorithm employs a piecewise linear prolongation operator. A possible explanation is that

the agglomeration algorithm creates better coarse grids than those employed in the non-nested algorithm. The CPU times required on the Cray Y-MP are 59 and 58 seconds with the original and the agglomerated multigrid, respectively, using three grids.

Perhaps the biggest advantage of the agglomeration algorithm lies in its ability to generate very coarse grids without any user intervention. Such extremely coarse grids should be beneficial in multigrid. Figure 10 shows two coarser grids for the four element airfoil case. These grids contain 63 and 22 vertices, respectively. With these grids it is now possible to use a 6 level agglomeration multigrid strategy. However, because these coarse grids are rather nonuniform, it is imperative that the first order coarse grid operator be a strictly positive scheme (i.e. one can no longer rely on assumptions of grid smoothness as conditions for stability). With the original first order operator in place, which is composed of a central difference plus a dissipative flux, it is difficult to guarantee the positivity of the scheme for arbitrary grids. In fact, the scheme has been found to be unstable on some of the very coarse and distorted agglomerated meshes. However, if the flux is replaced by a truly first order upwind flux, given for example by Roe's flux difference splitting [10], a stable scheme can be recovered for these coarse agglomerated grids. Thus, for each of the coarse grids obtained by agglomeration, a check of the convergence properties of the coarse grid operator at the desired flow conditions is carried out if problems are experienced with the multigrid. This step ensures that the coarse grid operators are convergent and that the problems with the multigrid, if any, come from the inter-grid communication. Figure 11 shows the convergence history with the 6 grid level agglomerated multigrid scheme. Also shown is the convergence with the 3 grid agglomeration multigrid scheme. In this particular case, Roe's upwind flux is used on the two coarsest grids, where central differencing proved unreliable. The time taken for the 6 grid agglomeration multigrid is 86 seconds. Thus the improved convergence rate is not entirely reflected in terms of the required computational resources. This is attributed to the increased time required by the Roe's upwind scheme, which involves a substantial number of floating point operations. This case serves to demonstrate the importance of the stability of each of the individual coarse grid operators in the multigrid scheme. Although first order upwinding has been employed on the distorted coarse meshes for demonstration purposes, it should be possible to construct stable central difference operators on such meshes.

5 Conclusions

It has been shown that the agglomeration multigrid strategy can be made to approximate the efficiency of the unstructured multigrid algorithm using independent, non-nested coarse meshes, in terms of both convergence rates and CPU times. It is further shown that arbitrarily coarse grids can be obtained with the agglomeration technique, although care must be taken to ensure that the coarse grid operator is convergent on these grids. Agglomeration has direct applications to three dimensions, where it may be difficult to derive coarse grids that conform to the geometry. In future work, alternate methods of generating coarse grids will be investigated. These may include the creation of maximal independent

sets to create the coarse grid seed points and using these seed points to agglomerate the fine grid cells around them. A maximal independent set is a subset of the graph containing only vertices that are distance 2 apart in the original graph. Since coarsening algorithms can be viewed as partitioning strategies, there also exists a possible interplay between agglomerated multigrid techniques and distributed memory parallel implementations of the algorithm, which should be further investigated. Finally, the implementation of the viscous terms for Navier-Stokes flows on arbitrary polygonal control volumes must be carried out for this type of strategy to be applicable to viscous flows.

6 Acknowledgements

The first author thanks the NAS Applied Research branch at NASA Ames Research Center for funding this project under contract NAS 2-12961. The second author's work was supported under NASA contract NAS1-19480. The third author was partially supported by AFOSR 91-0063, by DOE grant DE-FG02-88ER25053, by a PYI, NSF ASC-8858101 and by the Research Institute for Advanced Computer Science (RIACS) under Cooperative Agreement NCC2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

References

- [1] R.H. Ni. A Multiple Grid Scheme for Solving the Euler Equations *Proc AIAA Journal*, Vol 20, pp. 1565-1571, 1982.
- [2] A. Jameson. Solution of the Euler Equations by a Multigrid Method. In *Applied Math. and Comp.*, Vol. 13, pp. 327-356, 1983.
- [3] D. J. Mavriplis and A. Jameson. Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes. In *AIAA Journal*, Vol 26, No. 7, July 1988, pp. 824-831.
- [4] D. J. Mavriplis. Three Dimensional Multigrid for the Euler equations. *Proc AIAA 10th Comp. Fluid Dynamics Conf.*, Honolulu, Hawaii, June 1991, pp. 239-248.
- [5] M. P. Leclercq. Resolution des Equations d'Euler par des Methodes Multigrilles Conditions aux Limites en Regime Hypersonique. *Ph.D Thesis, Applied Math, Universite de Saint-Etienne*, April, 1990.
- [6] J. Peraire, J. Peiro, and K. Morgan. A 3D Finite-Element Multigrid Solver for the Euler Equations. *AIAA paper 92-0449*, January 1992.
- [7] M. Lallemand, H. Steve and A. Dervieux. Unstructured Multigridding by Volume Agglomeration: Current Status. In *Computers and Fluids*, Vol. 21, No. 3, pp. 397-433, 1992.

- [8] W. A. Smith. Multigrid Solution of Transonic Flow on Unstructured Grids. *Proc Recent Advances and Applications in Computational Fluid Dynamics, Proceedings of the ASME Winter Annual Meeting, Ed. O. Baysal*, November, 1990.
- [9] G. Spragle, W. A. Smith, and Y. Yadlin. Application of an Unstructured Flow Solver to Planes, Trains and Automobiles. *AIAA Paper 93-0889*, January 1993.
- [10] P., L. Roe., Approximate Reimann Solvers, Parameter Vectors, and Difference Schemes. In *Journal of Computational Physics*, Vol. 43, pp. 357-372, 1981.

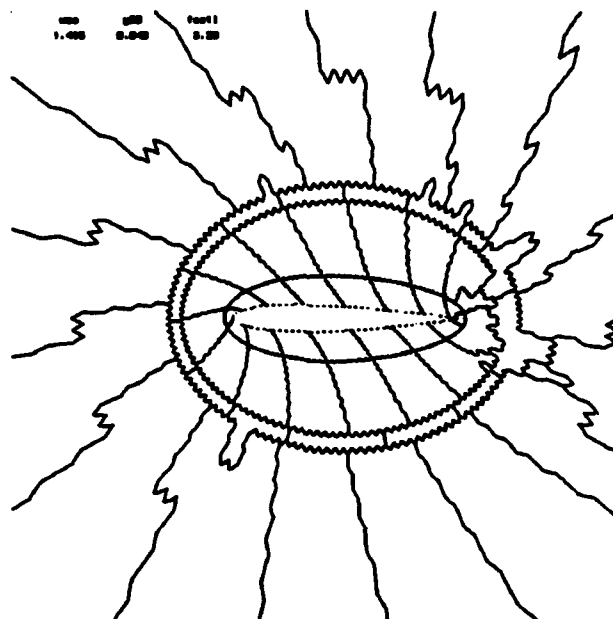
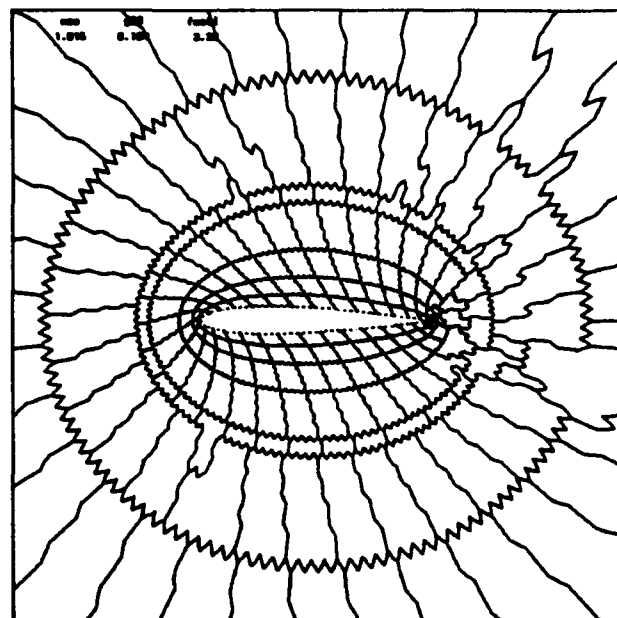
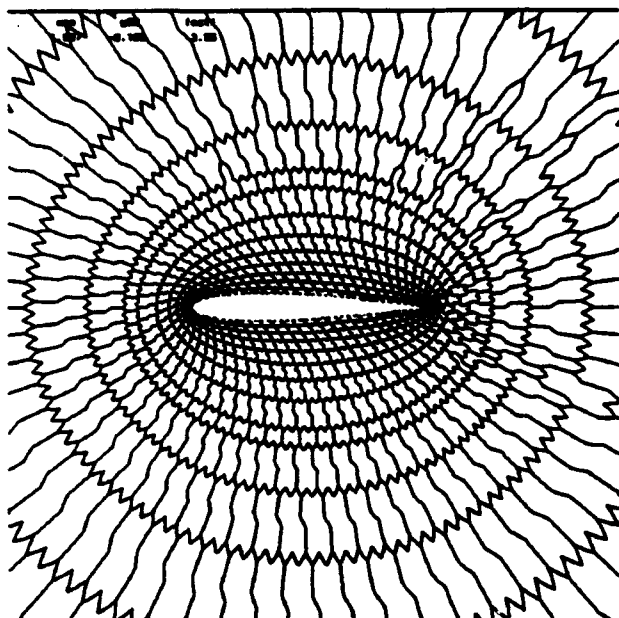


Figure 6: Three agglomerated coarse grids for the NACA0012 test case.

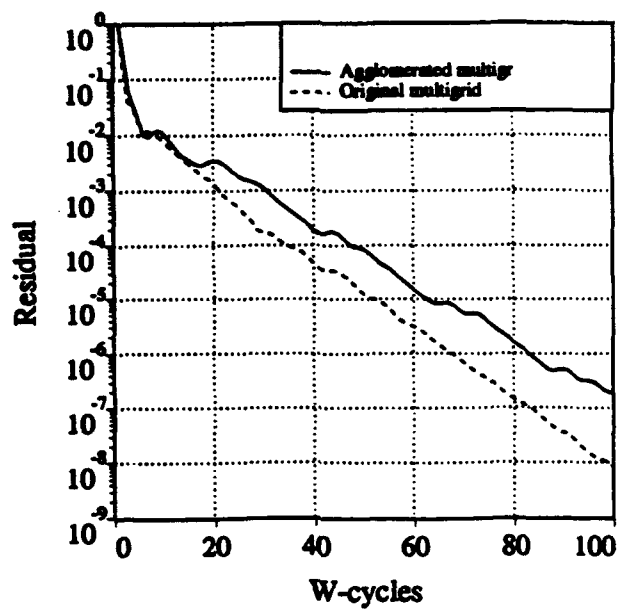


Figure 7: Convergence histories with the agglomerated and original multigrid.

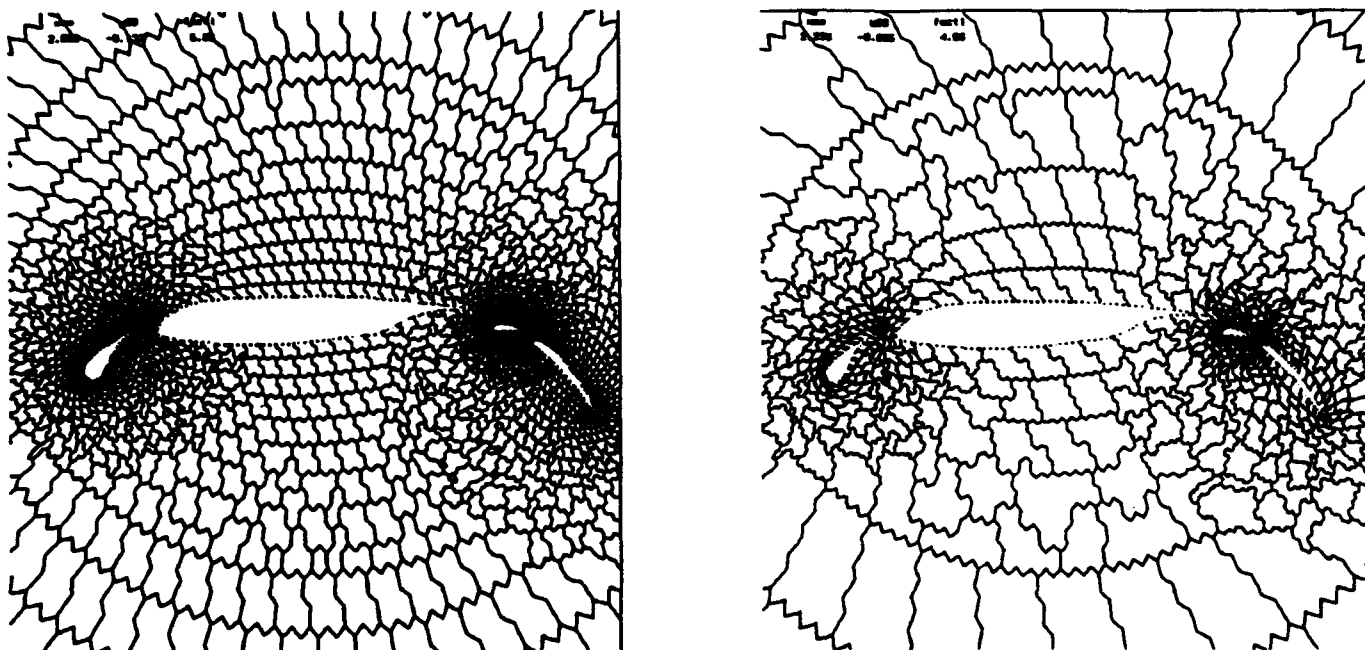


Figure 8: Two agglomerated coarse grids for the four-element test case.

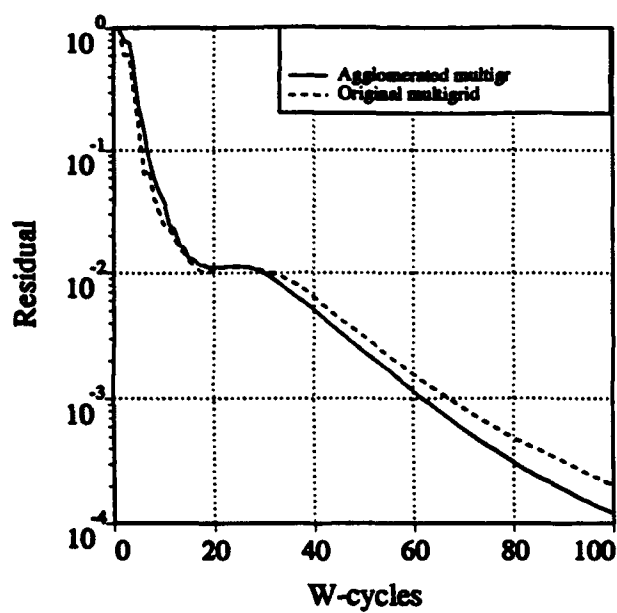


Figure 9: Convergence histories with the agglomerated and original multigrid.

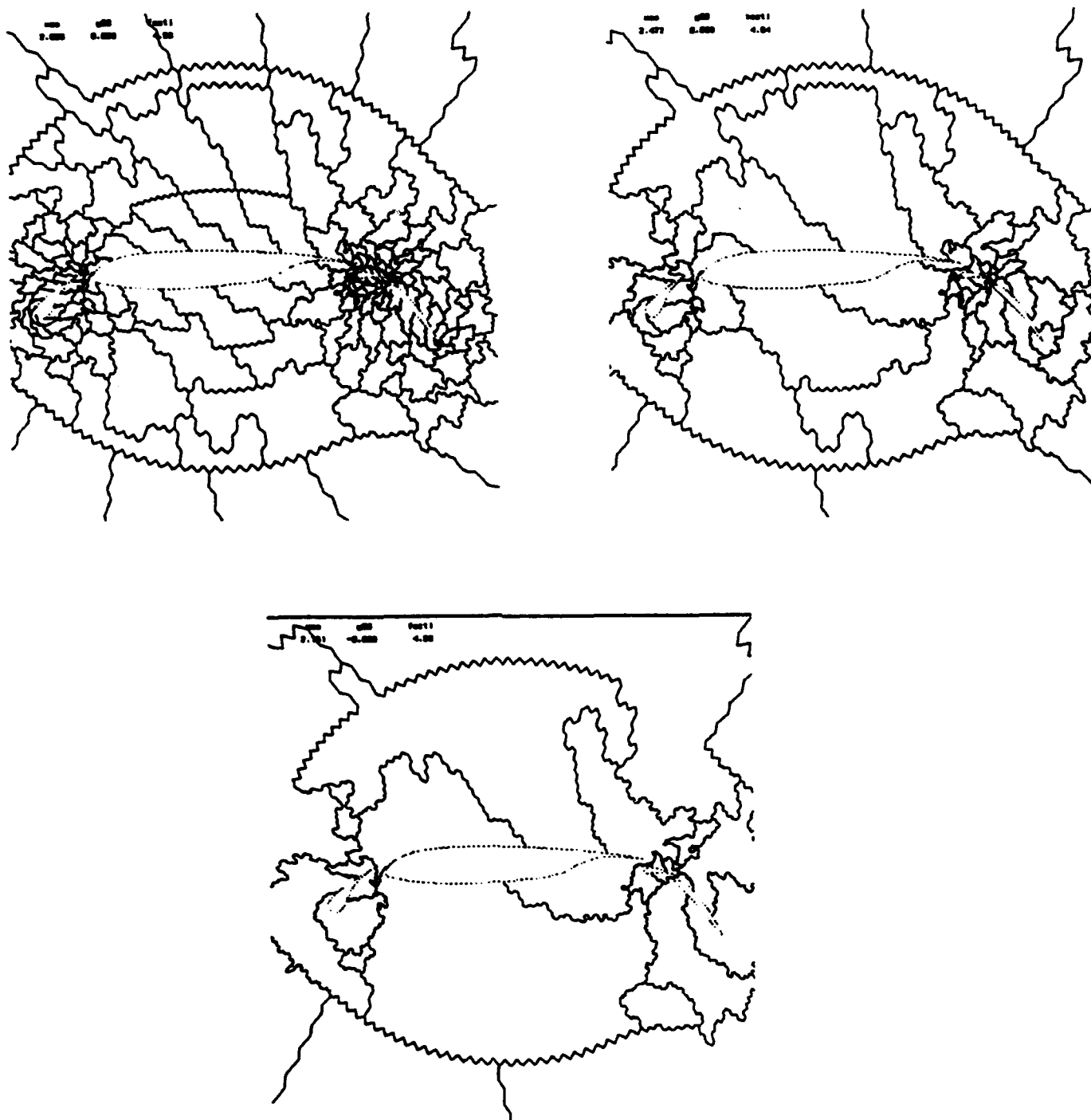


Figure 10: Three coarser grids for the four-element test case.

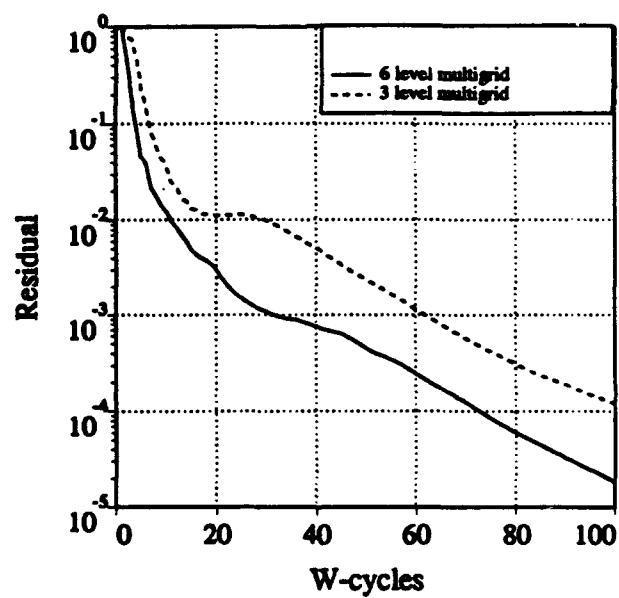


Figure 11: Convergence histories with the 6-level and 3-level agglomerated multigrid algorithms.

3D Automatic Cartesian Grid Generation for Euler Flows

John E. Melton
NASA Ames Research Center
Applied Aerodynamics Branch
Moffett Field, CA 94305

Francis Y. Enomoto
NASA Ames Research Center
Advanced Aerodynamics Concepts Branch
Moffett Field, CA 94305

Marsha J. Berger
Courant Institute
251 Mercer St.
New York, NY 10012

Abstract

We describe a Cartesian grid strategy for the study of three dimensional inviscid flows about arbitrary geometries that uses both conventional and CAD/CAM surface geometry databases. Initial applications of the technique are presented. The elimination of the body-fitted constraint allows the grid generation process to be automated, significantly reducing the time and effort required to develop suitable computational grids for inviscid flowfield simulations.

using a finite volume Cartesian grid-based flow solver, and makes possible the complete elimination of the surface gridding task and the automation of the volume grid generation.

There are currently three main approaches for dealing with the problem of complicated geometry. Body-fitted (structured) grids accommodate complex geometry using multiple blocks, and more recently, overlapping "composite" blocks as well [1, 2]. Although body-fitted grids currently produce the most accurate solutions, they are the most difficult to generate. This was the motivation for the second approach which uses unstructured body-fitted grids [3, 4]. Unstructured grids take advantage of the the complete geometric flexibility of triangles when defining the surface of the geometry. Unfortunately, they do not circumvent the labor-intensive task of generating acceptable surface triangulations, and require the generation of efficient, high-quality tetrahedral grids as well.

A third approach has recently been gaining popularity, namely, the use of non-body-

1 Introduction

Although the variety of complex aerospace geometries that can be analyzed with CFD continues to expand, the grid generation process remains both tedious and difficult. The purpose of this work is to demonstrate the advantages of integrating the CAD/CAM system into a fully automated grid generation and flow solver procedure. This approach is particularly straightforward when

fitted Cartesian grids [5, 6, 7, 8]. (There are at least 3 papers in this conference proceedings using this technique.) There are several reasons why the use of this technique should be further explored. These include the ease with which high order accurate integration schemes and multigrid acceleration can be implemented, and the relative geometric simplicity of the resulting grids. Cartesian grids can fairly easily incorporate an adaptive mesh refinement strategy to provide increased grid resolution. Perhaps the most widely known Cartesian grid method is found in TRANAIR, used routinely at Boeing and NASA Ames for the analysis of complete and complex configurations [9].

The most exciting reason to investigate the Cartesian grid approach is the ease with which a CAD/CAM-compatible geometry definition can be incorporated into an automated grid generation procedure. A CAD/CAM description of a collection of surfaces can be used directly in the computation of the geometric quantities needed for a flow solver using finite volume Cartesian grid cells. The surface modelling algorithms and software that are needed for these computations are typically proprietary and generally unavailable, but the source code for several modelling systems has recently become available, allowing this effort to proceed.

The drawbacks to the use of Cartesian grids stem primarily from the difficulty of imposing solid wall boundary conditions on a non-aligned grid. The geometry can intersect the grid in an essentially arbitrary way. Finite volume discretizations with sufficient accuracy are needed for the irregular cells of the Cartesian grid adjacent to a body component.

In this paper we describe a Cartesian mesh

algorithm for computing the inviscid flow-fields about complex geometries. We use the DTNURBS software library [10] to obtain the geometric quantities required for the finite volume flow field computations. Our main objective is to demonstrate the automatic grid generation procedure using Cartesian grids. An unstructured Cartesian grid flow solver (TIGER) previously developed by the author [11] was modified and used to integrate the Euler equations to steady state using Jameson's Runge Kutta timestepping algorithm with central differencing [12]. The modifications to the boundary conditions were only first order accurate; the next phase of this work will be to further develop the flow solver for more accurate solution on Cartesian meshes.

Section 2 of this paper describes the flow solver as it has been adapted for use with non-body-fitted Cartesian grids. Section 3 describes the geometry input definitions and the Cartesian grid generation algorithm. Computational results are presented in section 4. For a demonstration case, we compute the transonic flow about the ONERA M6 wing. We compare the results from CAD/CAM and faceted geometry input. We also include a more complex configuration (without the flow solution) to show the potential of this type of Cartesian grid representation. Conclusions are in section 5.

2 Cartesian Grid Flow Solver

The flow solver used in this report is a modified version of Jameson's four stage Runge Kutta algorithm for the solution of the Euler equations [12]. In integral form, the equa-

tions to be solved are

$$\frac{d}{dt} \iiint w \, dx \, dy \, dz = - \oint_S \mathbf{f} \cdot \mathbf{n} \, dS$$

where $w = (\rho, \rho u, \rho v, \rho w, \rho E)^t$ and

$$\mathbf{f} \cdot \mathbf{n} = \begin{pmatrix} \rho \mathbf{v} \cdot \mathbf{n} \\ \rho u \mathbf{v} \cdot \mathbf{n} + p n_x \\ \rho v \mathbf{v} \cdot \mathbf{n} + p n_y \\ \rho w \mathbf{v} \cdot \mathbf{n} + p n_z \\ (\rho E + p) \mathbf{v} \cdot \mathbf{n} \end{pmatrix}$$

These equations place no restrictions on the shape of an individual control volume. The main difference between a regular (non-body-intersecting) hexahedral cell and an intersected cell is the addition of a boundary term in the surface integrals which is used to impose the no normal-flux boundary conditions. The method uses central differencing in space, with second and fourth order dissipation added using a variable coefficient that is scaled by the local value of the second difference in pressure. At the irregular cells adjacent to the body, this scheme degenerates to first order differencing in space.

To implement the no-flux boundary condition, a surface area and a normal vector are required for the portion of the surface that lies within each intersected cell. We use the following observation to simplify the computation of the surface normal within each cell. The components of the surface normal vector can be obtained from the difference in exposed areas of opposing cell face pairs. This follows directly from the fact that the sum of the normals of each face multiplied by the area of the faces gives zero. Thus we do not explicitly compute the area of the intersection of the boundary surface within each cell. For simplicity, this is illustrated in two dimensions in figure 1. The x and y components of the surface normal vector are given

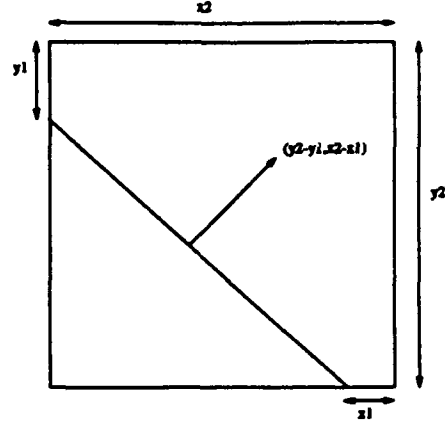


Figure 1: The normal vector can be computed from the exposed cell lengths $y1, y2$, and $x1, x2$.

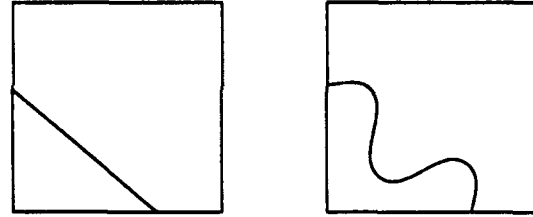


Figure 2: These two boundary surfaces have the same geometric description, i.e. surface normal and exposed cell face areas.

by the difference in the exposed areas $y2 - y1$ and $x2 - x1$, respectively. Note that in this formulation the flow solver can not tell the difference between the two surfaces shown in figure 2.

At the outer boundary, variables are extrapolated or specified depending on the Riemann invariants. Currently, the basic Jameson 4-stage Runge-Kutta algorithm with local time stepping is used to advance the solution. In future versions of the flow solver we will include a multigrid strategy to accelerate the convergence to steady state.

3 Automatic Cartesian Grid Generation

From the finite volume formulation of the Euler equations, the geometric information needed for each control volume includes

- Area of each cell face
- Direction of outward normal vector for body surface
- Centroids of cell faces and exposed cell volume (for second order schemes).

If steady state rather than time accurate solutions are required, the exposed cell volume is not actually necessary, and can be replaced by the hexahedral volume in the time-stepping scheme.

The automated grid generation techniques described in this report accept two basic surface geometry input formats. For the first, we use the NURBS format (Non-Uniform Rational B-Spline), used in most modern CAD/CAM systems as the typical entity for geometry description [13]. NURBS are able to represent complicated curved geometries with a relatively small number of control parameters, and provide a complete description of the surface and its derivatives. One difficulty in the use of NURBS is reflected in the algorithms required for their interrogation. For example, the calculation of NURBS surface-surface intersections typically requires finding all solutions to a high order nonlinear polynomial equation, thus requiring an iterative procedure and the specification of a root-finding tolerance. An additional difficulty is the lack of any guarantee that all intersections will be found. A probability factor is used to control the

amount of searching performed by the root-finding subroutines. Despite these complications, there is no reason why these algorithms cannot be used in an approach that eliminates the tedious and time consuming tasks of interactive surface and volume grid generation while retaining the complete geometric accuracy of the NURBS surface definition.

We have chosen to use the DTNURBS collection of computational geometry routines because of the availability of the FORTRAN source code, but many of the other proprietary packages contain essentially the same functionality. One limitation of the current version of DTNURBS is the lack of routines capable of operating on trimmed surfaces, so all of the geometries discussed in this report were composed of multiple natural surfaces. The creation of these NURBS surfaces is the only part of the grid generation procedure that requires human intervention.

Our goal then is to compute the finite volume cell geometric information directly from the NURBS description of a geometry obtained from the CAD/CAM system via the IGES file format [14]. The approach eliminates the need to generate a surface discretization before a volume (flowfield) grid can be created. We generate the Cartesian volume grid in two steps. We begin by creating a coarse, equi-spaced mesh of cells. Each cell face is then checked for an intersection with the surface. For computational efficiency, this step is performed in three stages. In the first stage, the edges of each face are checked for surface intersections. If none are found, each individual grid cell face is then converted into a NURBS description and input to the surface-surface intersection routine. This second stage attempts to detect any intersections between

the surface NURBS geometry and the interior of the cell face. If no intersections are found in the first two stages, a third stage is used to determine if the face is wholly internal or external to the surface. We proceed in this way since the surface-surface intersection algorithm is the least efficient of the NURBS interrogation routines.

Local refinement for the purpose of geometry definition is done as the grid is generated. If a cell needs to be refined, it is done during the first step immediately after processing the parent cell. This saves some computational expense. For example, after computing intersections, we mark non-intersecting cell faces with a flag that denotes them as fully internal or fully external. If the cell is then refined, the children cells inherit this property and need not be further examined.

After the cell vertices of the initial refined grid have been established, the second step is the calculation of the face areas for those cells that intersect the surface. DTNURBS does not currently provide the capability needed to do this with a single subroutine call. Instead, we proceed in an indirect manner by first computing the spline curve describing the intersection of the plane containing the cell face and surface geometry. This spline is then converted into a piecewise linear curve describing the body cross section. This conversion is done with high accuracy using a curvature-sensitive DTNURBS subroutine. Finally, the area of the portion of the cell face exterior to the body is computed. (The details of this last step are explained in the discussion of the triangular geometry input format).

When used with the NURBS geometry, we currently refine all cells that intersect the surface geometry to a maximum prescribed

level. However, the DTNURBS package provides a subroutine that calculates curvature at a given point. Our plan is to use this (and other) geometric refinement criterion in conjunction with flowfield criteria in order to develop a fully automated grid refinement procedure applicable to arbitrary geometries and flow conditions.

The second geometry input format that this program accepts is the more familiar one consisting of a collection of triangles describing the surface of the geometry. The only requirements for the surface triangulation are that it not contain any zero-thickness components and that it be watertight, i.e., all edges of each triangle must be matched by the edge of another triangle. The intersections between the Cartesian grid cells and the body triangles and the amount of face area external to the geometry can then be determined using well known planar computational geometry algorithms. With careful programming, many of these geometrical computations can be vectorized, resulting in an efficient and automated Cartesian grid generation algorithm for arbitrarily shaped triangulated geometries.

The generation of the refined Cartesian grid proceeds using the same two steps described previously. First, each cell face is checked for intersection with the triangular facets that compose the surface. Since both the cell face and triangle are planar polygons this is a simple operation: each edge of each polygon is checked for an intersection within the interior of the other polygon. In the second step, the cell areas exterior to the geometry are computed in the following manner. First, a planar cross-section of the surface triangulation is computed, yielding a collection of line segments coplanar with an intercepted

cell face. These segments are then ordered and joined into closed polygons describing the cross-section. Finally, the Sutherland-Hodgman polygon clipping algorithm [15] is used to determine the portions of the cross-section polygons that lie within each rectangular cell face.

Because of the general nature of the surface geometry, the cross sections that result from a planar slice can intersect the cell in an arbitrary manner. For example, the cross section of an engine nacelle located entirely within the face of a coarse grid cell is shown in figure 3. One indication that further cell refinement is needed is given by the existence of multiple independent regions (such as A and B) within a cell. Care must therefore be taken to determine the topology of the cross sections, including those composed of multiple independent and/or concentric polygons, in order that decisions about additional cell refinements can be made automatically. One step of this topology determination process is illustrated in figure 3. We determine whether the area enclosed within each cross-section polygon is interior or external to the geometry by casting a ray emanating from a point on the cross-section and counting the number of intersections that the ray makes with other cross sections. If this number is even, the region enclosed by the contour must be internal to the geometry.

4 Computational Examples

We show three flow solutions computed for the ONERA M6 wing. The final example illustrates a grid generated for a complex configuration without flow solution.

In figure 4, we show C_p distributions for the ONERA M6 wing at the standard test

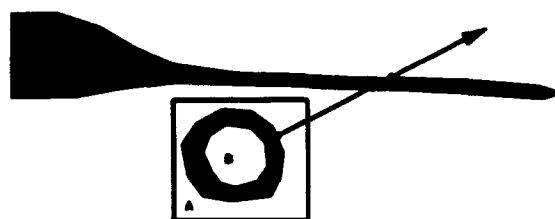


Figure 3: Illustration of the intersection-counting procedure for cross sections.

conditions of Mach 0.84 and 3.06 degrees angle of attack [16]. Two coarse grids of equivalent density were generated using the NURBS and triangle input geometry formats. The NURBS surface input file contained four surfaces. The tip and the trailing edge thickness were modeled with single surfaces, and the remainder of the wing was split into two NURBS that defined the upper and lower surfaces. An additional fine mesh was generated using the triangle input file. Figure 5 shows the improvements in the C_p distributions obtained on this fine grid. Figure 6 shows the fine grid and Mach number distribution along the centerline symmetry plane and at two outboard wing stations. The various levels of surface grid refinement are also evident, and correspond to the different colorings of the wing surface triangles. These figures demonstrate that the lack of agreement between the NURBS solution and the experimental data results from inadequate flowfield resolution (especially near the leading edge), and not from any inaccuracies in the NURBS geometrical computations. The two coarse grids were created on a Silicon Graphics Indigo Elan workstation. For the NURBS definition of the wing, the complete grid generation process required approximately four hours for a mesh of 20,804 cells, 3,286 of which in-

tersected the body geometry. The corresponding triangle geometry grid had 20,814 cells, but was generated in 5 minutes. The high resolution grid was generated on the NAS Cray C90, and required 124 seconds to produce a total of 134,198 cells, 11,402 of which intersected the surface. The flow solver was run for each of the grids on the NAS Cray C90, and executed at a rate of 8.8 microseconds/cell/timestep. All of the solutions shown were converged to four orders of magnitude in the density residual. Figures 7 and 8 show two views of a Cartesian grid generated using a quadrilateral description of an F16XL. Each quadrilateral was decomposed into two triangles before being input to the grid generator. The resulting grid contained eight levels of refinement, with a total of 277,657 cells in the flowfield and 56,205 surface-intersecting cells. The grid required 375 seconds to generate on the Cray C90.

5 Conclusions

The use of Cartesian grids generated directly from the CAD/CAM surface definition makes possible an automated grid generation procedure applicable to arbitrary three dimensional configurations. We have demonstrated the geometric capability; the next step is improvement in the numerical algorithm for the flow simulation. This approach shows promise for dramatically reducing the time required to produce accurate CFD simulations about complicated vehicles.

Acknowledgements

M. Berger was supported in part by DOE grant DE-FG02-88ER25053, by AFOSR

grant 91-0063, and by a PYI, NSF ASC-8858101.

References

- [1] J. Thompson. *Numerical Grid Generation*. North-Holland, 1982.
- [2] J. Steger, Benek, and Dougherty. A Flexible Grid Embedding Technique With Application to the Euler Equations. *AIAA-89-1944*, July 1983. 6th Computational Fluid Dynamics Conf., Danvers, Mass.
- [3] D. J. Mavriplis and A. Jameson. Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes. In *Proc. Third Copper Mountain Conf. Multigrid Methods, Lecture Notes in Pure and Applied Mathematics*, 1987. ICASE Report No. 87-53.
- [4] J. D. Baum and R. Löhner. Numerical Design of a Passive Shock Deflector Using an Adaptive Finite Element Scheme on Unstructured Grids. *AIAA-92-0448*, 1992.
- [5] R. Gaffney, H. Hassan, and M. Salas. Euler Calculations for Wings Using Cartesian Grids. *AIAA-87-0956*, January 1987.
- [6] M.J. Berger and R.J. LeVeque. An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries. *AIAA-89-1990*, June 1989. 9th Computational Fluid Dynamics Conf., Buffalo, NY.
- [7] D. DeZeeuw and K. Powell. An Adaptively Refined Cartesian Mesh Solver for the Euler Equations. *AIAA-91-1542*, 1991.
- [8] B. Epstein, A. Luntz, and A. Nachschon. Cartesian Euler Method for Arbitrary Aircraft Configurations. *AIAA Journal*, 30(3):679-687, March 1992.
- [9] D. Young, R. Melvin, M. Bieterman, F. Johnson, S. Samant, and J. Bussioletti. A Locally Refined Rectangular Grid Finite Element Method: Application to Computational Fluid Dynamics and Computational Physics. *J. Comp. Phys.*, January 1991.

- [10] Boeing Computer Services. The DT-NURBS Spline Geometry Subprogram Library User's Manual, Version 2.0. Technical report, October 1992.
- [11] J. Melton, S. Thomas, and G. Cappuccio. Unstructured Euler Flow Solutions Using Hexahedral Cell Refinement. *AIAA-91-0637*, January 1991. 29th Aerospace Sciences Mtg., Reno, Nevada.
- [12] A. Jameson, W. Schmidt, and E. Turkel. Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes. *AIAA-81-1259*.
- [13] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design, a Practical Guide*. Academic Press, 1988.
- [14] Kent Reed. The Initial Graphics Exchange Specification (IGES) Version 5.1. September 1991.
- [15] Foley, van Dam, Feiner, and Hughes. *Computer Graphics, Principles and Practice*. Addison Wesley, 1990.
- [16] V. Schmitt and F. Charpin. Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers. AGARD Report AR-138, 1979.

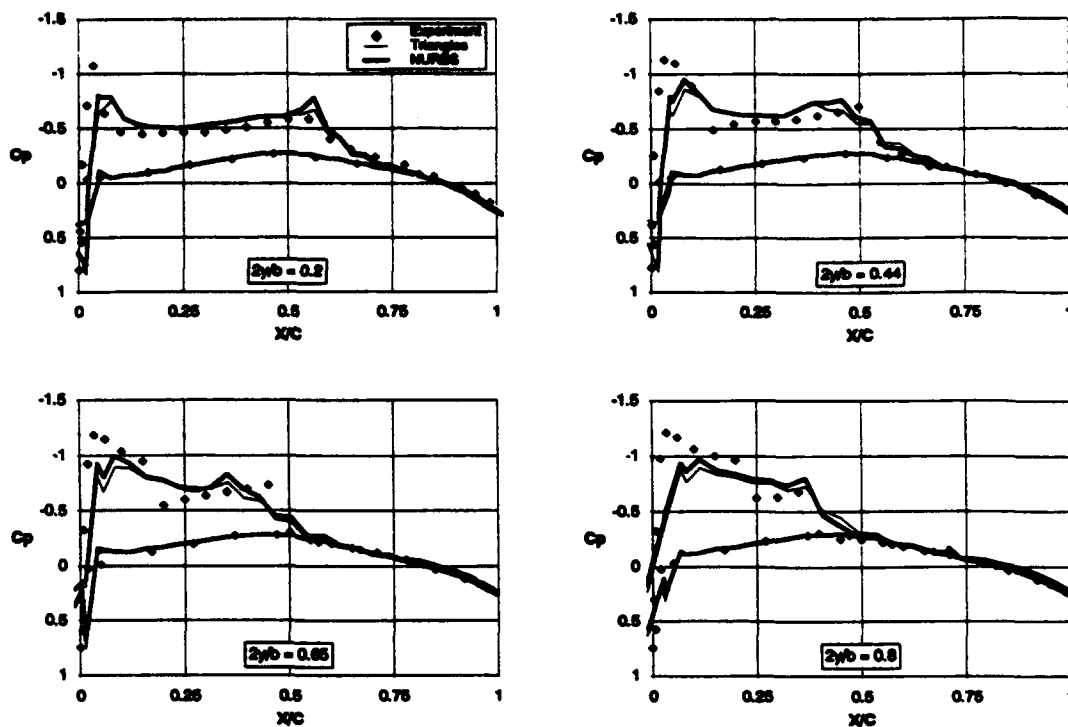


Figure 4: Coarse grid C_p distributions for the ONERA M6 wing, Mach 0.84, $AoA = 3.06$

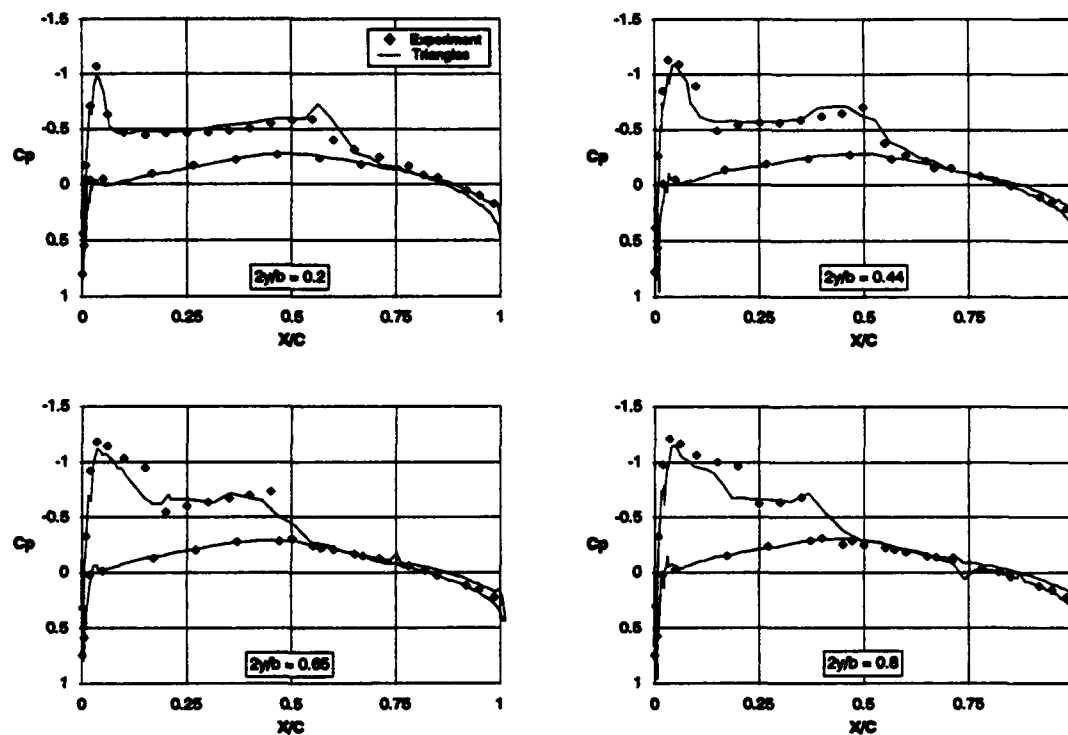


Figure 5: Fine grid C_p distributions for the ONERA M6 wing, Mach 0.84, $AoA = 3.06$

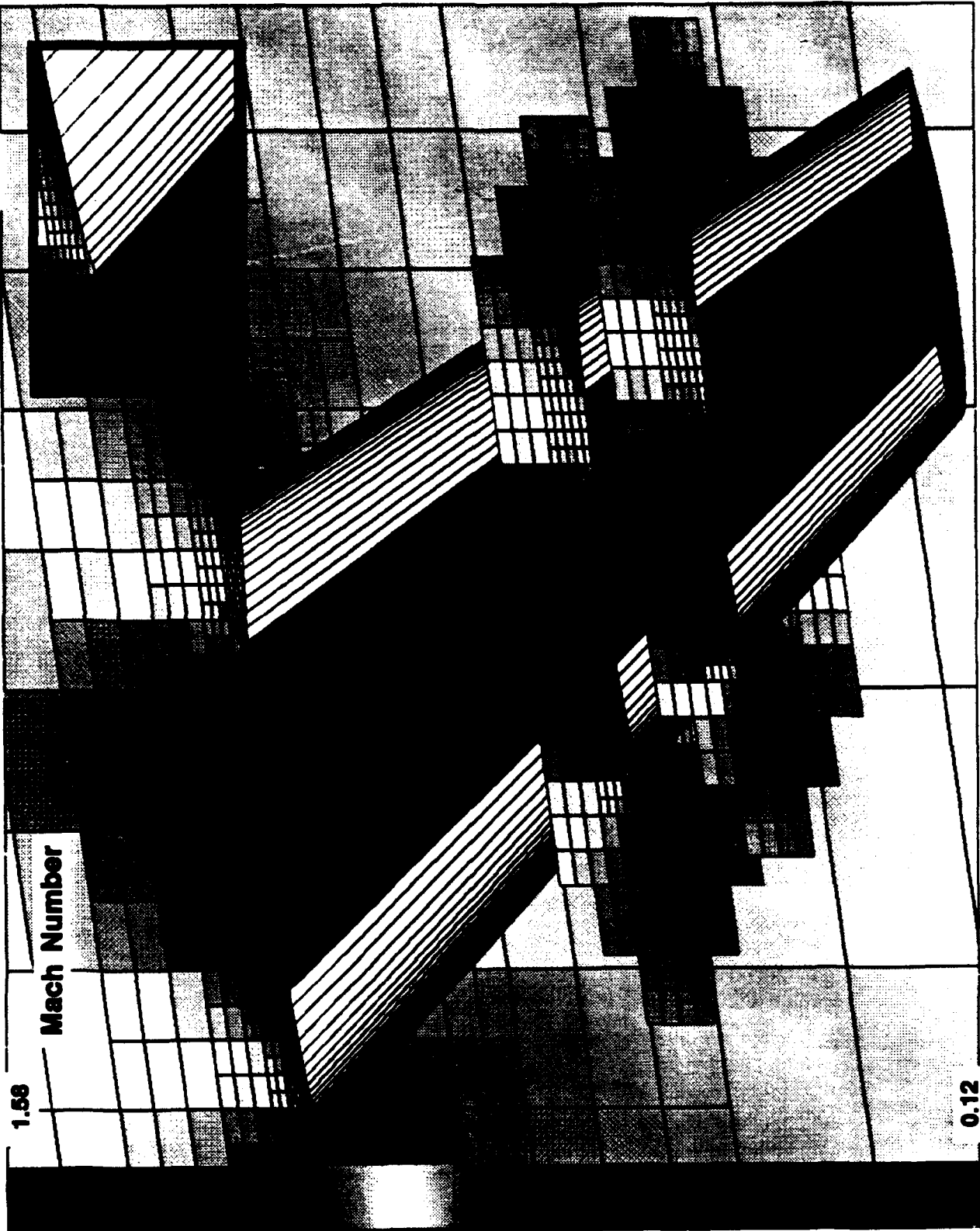


Figure 6: Isometric view of ONERA M6 wing (inset shows leading edge detail)

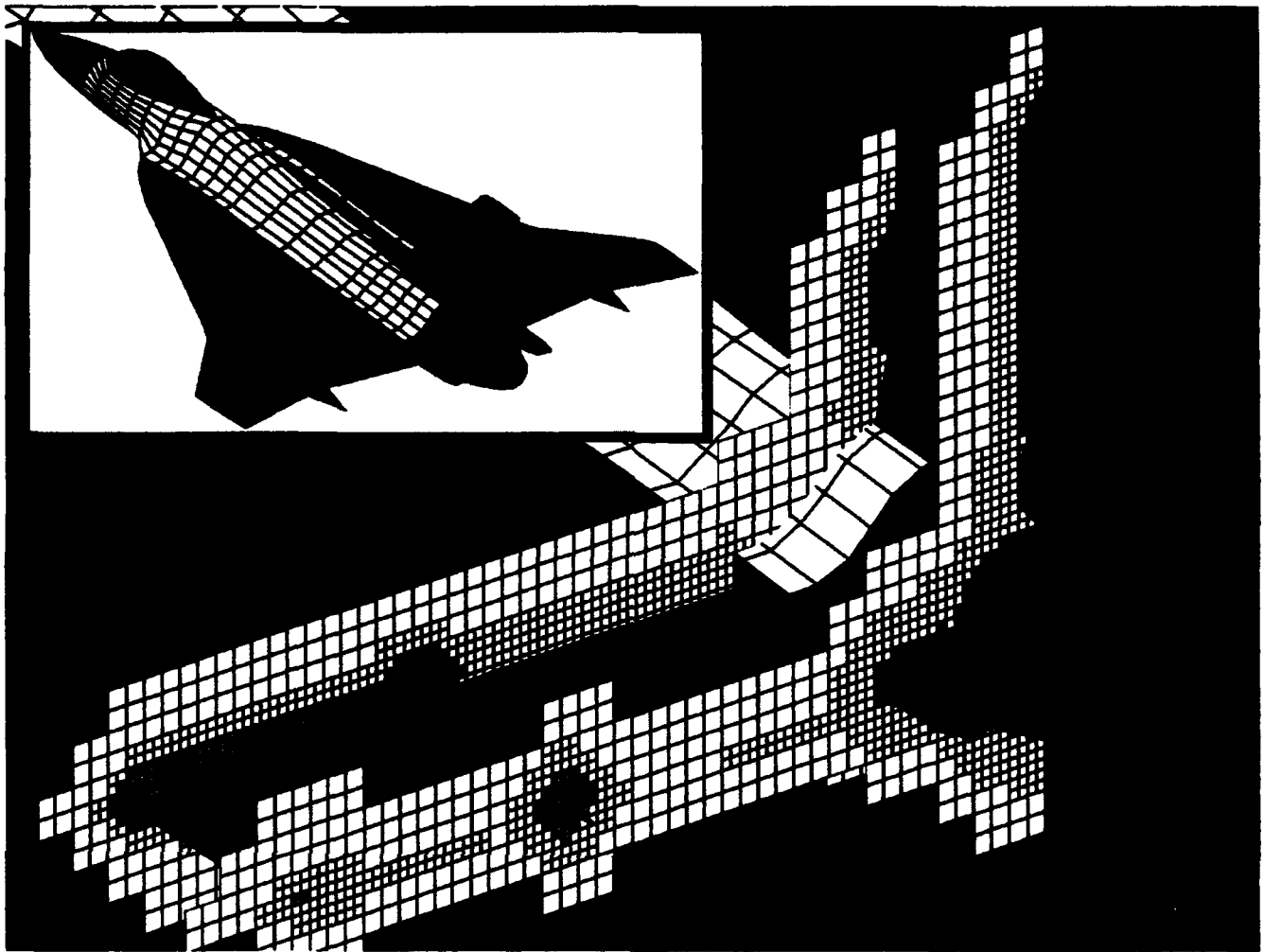


Figure 7: Aft view of F16XL Cartesian Grid (277657 cells)

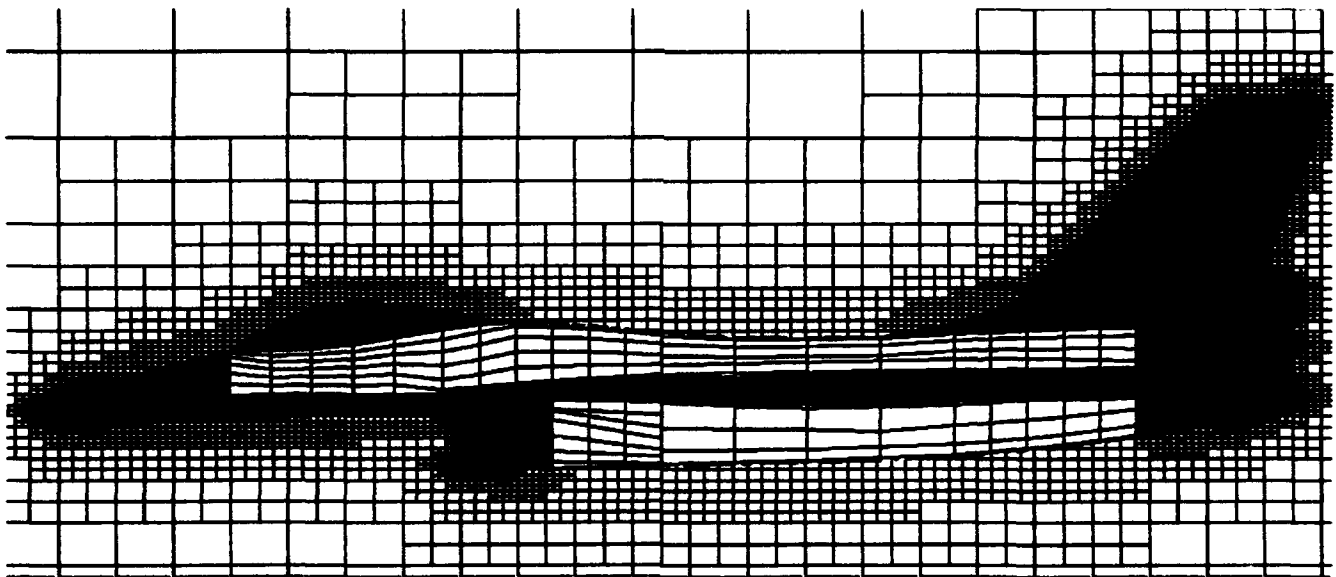


Figure 8: Side view of F16XL Cartesian Grid

Three Dimensional Adaptive Mesh Refinement for Hyperbolic Conservation Laws

John Bell

Lawrence Livermore Laboratory
Livermore, CA 94550

Marsha Berger

Courant Institute, NYU
New York, NY 10012

Jeff Saltzman

Los Alamos National Laboratory
Los Alamos, NM 87545

Mike Welcome

Lawrence Livermore Laboratory
Livermore, CA 94550

ABSTRACT

We describe a local adaptive mesh refinement algorithm for solving hyperbolic systems of conservation laws in three space dimensions. The method is based on the use of local grid patches superimposed on a coarse grid to achieve sufficient resolution in the solution. A numerical example computing the interaction of a shock with a dense cloud in a supersonic inviscid regime is presented. We give detailed timings to illustrate the performance of the method in three dimensions.

1. Introduction

Advanced finite difference methods, by themselves, are unable to provide adequate resolution of three dimensional phenomena without overwhelming currently available computer resources. High-resolution 3D modeling requires algorithms that focus the computational effort where it is needed. In this paper we extend the Adaptive Mesh Refinement (AMR) algorithm for hyperbolic conservation laws originally developed in [1] to three space dimensions. AMR is based on a sequence of nested grids with finer and finer mesh spacing in both time and space. These fine grids are recursively embedded in coarser grids until the solution is sufficiently resolved. An error estimation procedure automatically determines the accuracy of the solution and grid generation procedures dynamically create or remove rectangular fine grid patches. Special difference equations are used at the interface between coarse and fine grid patches to insure conservation. This is all handled without user intervention by the AMR program.

Two dimensional versions of the AMR algorithm described here have been used to solve fluid flow problems in a variety of settings, and has enabled the study of fluid flow phenomena not previously possible. For example, the extra resolution provided by AMR enabled the computation of a Kelvin-Helmholtz type instability along the slip line in a study of Mach reflection off an oblique wedge [2], and aided in the resolution of the weak von Neumann paradox in shock reflection [3]. When combined with a multifluid capability, the algorithm was used to compute the interaction of a supernova remnant with an interstellar cloud [4], and to categorize refraction patterns when a shock hits an oblique material interface [5]. When extended to use body-fitted coordinates, AMR was used to study diffraction of a shock over an obstacle [6]. In each of these cases the use of adaptive mesh refinement reduced the cost of the computation by more than an order of magnitude. The improved efficiency associated with using AMR may make similar flows in three dimensions computationally tractable.

There are several alternative approaches to focusing computational effort in three-dimensional flows. One approach uses a logically rectangular grid with moving grid points that adjust to the flow. There are several drawbacks to this approach. First, it is hard to implement a three dimensional high-resolution integration scheme for moving rectilinear grids. In our approach the integration scheme need only be defined for uniform rectangular grids; this avoids the complexity and computational cost associated with metric coefficients in the moving grid approach. Furthermore, in three dimensions it is extremely difficult to effectively cluster points to capture unsteady phenomena while maintaining a grid with sufficient smoothness in both space and time to permit effective computation. Even if acceptable grid motion can be determined, the entire computation is usually performed with a fixed number of zones throughout the computation. The local grid refinement approach dynamically adjusts the number of zones to match the requirements of the computation. The time step used in moving mesh codes is also limited by the smallest cell size unless additional work is done by solving the equations implicitly or using techniques that allow each cell to evolve with its own time step.

Another approach to three dimensional computations uses adaptive unstructured grids. Unstructured grids offer the most flexibility in optimally placing zones; however, we favor locally uniform patches for their accuracy and wave propagation properties. The development of discretization techniques that avoid degradation for strong shocks on highly irregular meshes remains an open issue. Our use of uniform grids allows us to directly use much of the high resolution difference scheme methodology developed for this flow regime. Uniform patches also have low overhead, both from the computational and the storage point of view. The extra information that is needed, in addition to the actual solution values, is proportional to the number of grids rather than the total number of grid points. Scratch space needed during integration is also reduced by using uniform grids. The additional storage of AMR is negligible.

An indication of the robustness of the mesh refinement algorithm is that very few changes were required in extending it from two to three dimensions. However, time-dependent three-dimensional computations push the limits of current machine resources, both in terms of memory and CPU time. For this

reason, particular care was taken in the implementation, and the grid generation algorithm was improved to increase the overall efficiency of the code.

The starting point for this paper is the version of AMR presented in [2] and we assume the reader is familiar with that paper. In section 2 we describe the differences in the three dimensional mesh refinement algorithm which have to do with the grid generation algorithm and the error estimator. In section 3 we describe the operator split integration scheme used in the numerical experiment. In particular, this section clarifies the interaction of the grid refinement and operator split boundary conditions on patched grids while maintaining conservation at grid interfaces. We include a brief section on implementation since some simple changes that produce a much cleaner code have been incorporated. We are also rewriting the code in C++. The results of a numerical experiment of a shock-cloud interaction modeling the laboratory experiments of Sturtevant and Haas [7] are presented in section 5. Detailed timings are presented as well as memory usage and grid statistics demonstrating that AMR offers significant savings of computational resources and can be an important tool in the study of three dimensional fluid dynamics.

2. The Adaptive Mesh Refinement Algorithm

AMR uses a nested sequence of logically rectangular meshes to solve a PDE. In this work, we assume the domain is a single rectangular parallelepiped although it may be decomposed into several coarse grids. With the new grid generator described below, *grids at the same level of refinement do not overlap*. We require that the discrete solution be independent of the particular decomposition of the domain into subgrids. Grids must be properly nested, i.e. a fine grid should be at least one cell away from the boundary of the next coarser grid unless it is touching the boundary of the physical domain. However, a fine grid can cross a coarser grid boundary and still be properly nested. In this case, the fine grid has more than one parent grid. This is illustrated in Figure 1 in two dimensions. (This set of grids was created for a problem with initial conditions specifying a circular discontinuity).

AMR contains five relatively separate components. The *error estimator* uses Richardson extrapolation to estimate the local truncation error; this determines where the solution accuracy is insufficient. The *grid generator* creates fine grid patches covering the regions needing refinement. *Data structure* routines manage the grid hierarchy allowing access to the individual grid patches as needed. *Interpolation* routines initialize a solution on a newly created fine grid and also provide the boundary conditions for integrating the fine grids. *Flux correction* routines insure conservation at grid interfaces by modifying the coarse grid solution for coarse cells that are adjacent to a fine grid.

When all these components are assembled, a typical integration step proceeds as follows. The integration steps on different grids are interleaved so that before advancing a grid all the finer level grids have been integrated to the same time. One coarse grid cycle is then the basic unit of the algorithm. The variable r denotes the mesh refinement factor in both space and time (typically 4), and *level* refers to the number of refinements (the coarsest grid is at level 0). The regridding procedure is done every few steps,

so any particular step may or may not involve regridding.

Recursive Procedure Integrate (*level*)

Repeat r_{level} times

Regridding time? - error estimation and grid generation for *level* grids and finer
step Δt_{level} on all grids at level *level*

if (*level* + 1) grids exist

then begin

integrate (*level* + 1)

conservation_fixup(*level*, *level* + 1)

end

end.

level = 0 (* coarsest grid level *)

Integrate (*level*)

Single Coarse Grid Integration Cycle

All of these steps are described fully in [2], with the exception of the grid generation algorithm.

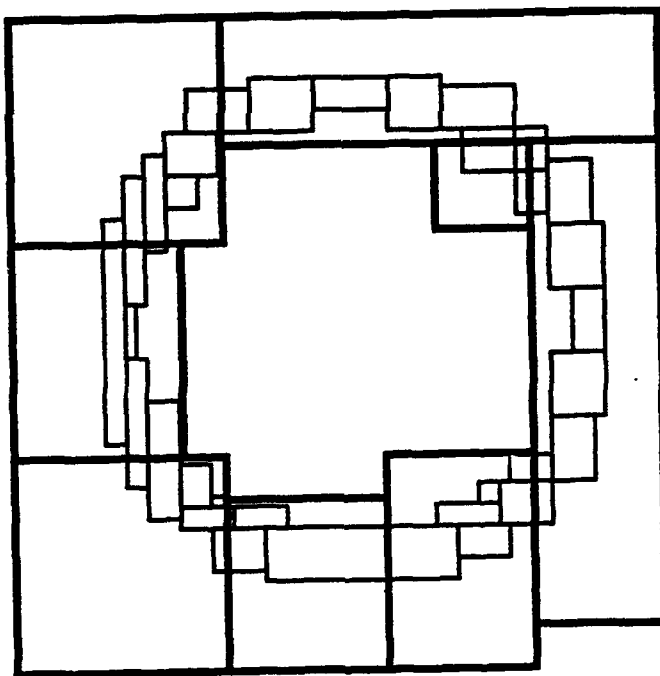


Figure 1 illustrates a coarse grid with two levels of refined grids. The grids are properly nested, but may have more than one parent grid.

2.1. Grid Generation

The grid generation algorithm takes a list of coarse grid points tagged as needing refinement and groups them in clusters. Fine grids are then defined by fitting the smallest possible rectangles around each cluster. The objective of the grid generation is to produce efficient grids, i.e. rectangles containing a minimum number of cells that are not tagged, without creating a large number of small grids with poor vector performance and excessive boundary overhead.

The grid generator in [2] uses a simple bisection algorithm. If a single enclosing rectangle is too inefficient, it is bisected in the long direction, the tagged points are sorted into their respective halves, and new enclosing rectangles are calculated. The efficiency is measured by taking the ratio of tagged points to all points in a new fine grid. The procedure is repeated recursively if any of the new rectangles are also inefficient. Since this algorithm uses no geometric information from the tagged points, it often results in too many tiny subgrids and is followed by a merging step. Unfortunately, this results in overlapping grids. Since the memory usage in three dimensional calculations is at a premium, we want to avoid overlapping. Furthermore, we expect that there will be large numbers of grids in three dimensions which makes the merging step costly.

We have developed a new clustering algorithm that uses a combination of signatures and edge detection. Both techniques are common in the computer vision and pattern recognition literature. After much experimentation, described in [8], we have developed what amounts to a "smart bisection" algorithm. Instead of cutting an inefficient rectangle in half, we look for an "edge" where a transition from a flagged point region to a non-flagged one occurs. The most prominent such transition represents a natural line with respect to which the original grid can be partitioned.

We describe the procedure in two dimensions for purposes of illustration. First, the signatures of the flagged points are computed in each direction. Given a continuous function $f(x, y)$, the horizontal and vertical signatures, Σ_x and Σ_y , are defined as

$$\Sigma_x = \int f(x, y) dy$$

and

$$\Sigma_y = \int f(x, y) dx$$

respectively. For discrete binary images, this is just the sum of the number of tagged points in each row and column. If either signature contains a zero value, then clearly a rectangle can be partitioned into two separate clusters in the appropriate direction. If not, an edge is found by looking for a zero crossing in the second derivative of the signature. If there is more than one such zero crossing, the largest one determines the location for the partitioning of the rectangle. If two zero crossings are of equal strength, we use the one closest to the center of the old rectangle to prevent the formation of long thin rectangles with poor vectorization. This procedure is also applied recursively if the resulting rectangles do not meet the efficiency

criterion, with the exception that if no good partition is found, and the efficiency is at least 50%, the rectangle is accepted; otherwise it is bisected in the long direction as a last resort. In computational experiments in two space dimensions, for the same level of accuracy the new algorithm reduces the CPU time by approximately 20%.

Figure 2 illustrates the entire procedure on a sample set of points. The first column on each side is the signature, and next to it is the second derivative. After each partitioning of the points, a new enclosing rectangle is calculated around the tagged points. In this figure, after 3 steps, the fine grids are acceptably efficient and the procedure stops.

2.2. Error Estimation

The second improvement over the basic approach in [2] is the addition of a purely spatial component to the error estimation process to supplement Richardson extrapolation. In Richardson extrapolation, the data on the grid where the error is being estimated is coarsened and then integrated for a timestep. That result is then compared to the result of integrating first and then coarsening. For smooth solutions, the difference in these two results is proportional to the truncation error of the scheme. The motivation for including an additional error measure is to identify structures that are missed by the averaging process associated with the coarsening in the Richardson extrapolation. For example, in gas dynamics a slow-moving or stationary contact surface generates little or no error in the Richardson extrapolation process. In fact, the integrator is not generating any error in this case. However, failure to tag the contact will cause it to be smeared over a coarser grid. The error associated with deciding whether or not to refine the contact surface is associated only with the spatial resolution of the discontinuity, not with errors in integration. Should the contact need to be refined later, (for example if it interacts with another discontinuity), the initial conditions are no longer available to provide higher resolution. For certain special cases a similar phenomenon can also occur for shocks. These problems can be avoided by providing the error estimation routine with the unaveraged grid data so that spatial resolution can also be measured.

Along with the addition of a purely spatial component to the error estimation, we also directly control the process of tagging (and untagging) cells for refinement. For example, a user can insist that only a certain part of the domain is of interest and that the error estimator should be ignored if it says refinement is needed outside of the interesting regions. Similarly, it can force refinement in a particular region independent of the error estimation result.

3. Integration Algorithm

For the computational examples presented in section 5, we use an operator-split second-order Godunov integration scheme. However, the particular form of the integration scheme is independent of the remainder of the AMR shell. Other integration methods and, in fact, other hyperbolic systems can be easily inserted into the overall AMR framework. The only requirement for the integration scheme is that it

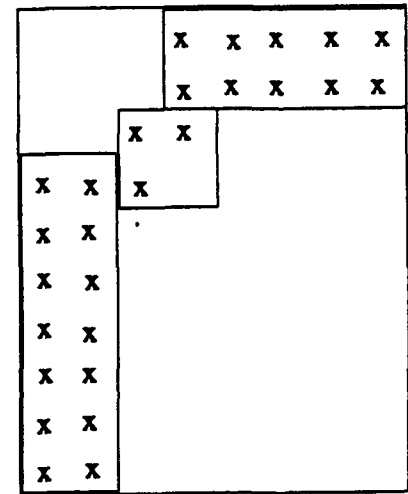
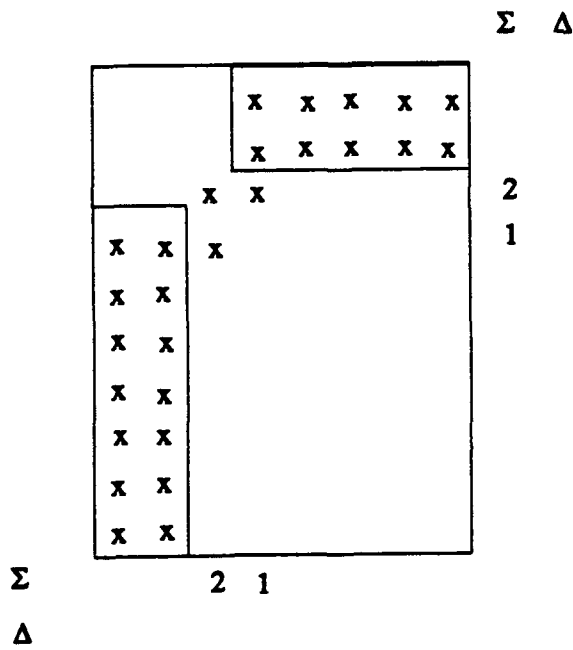
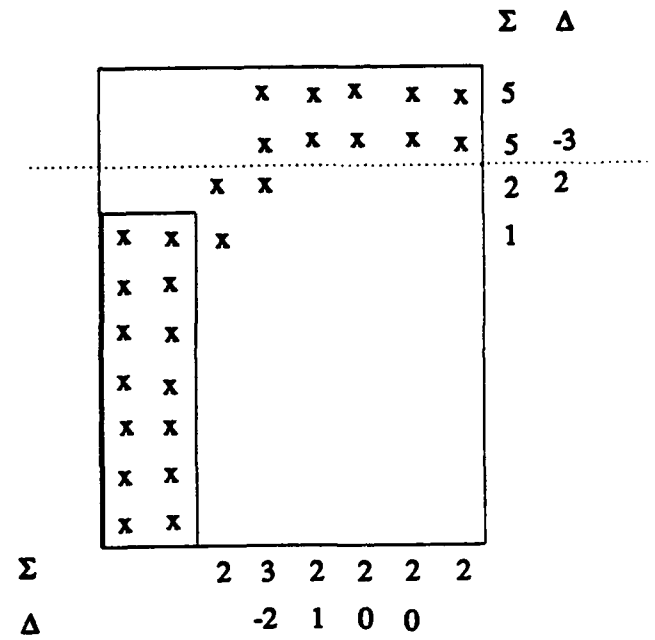
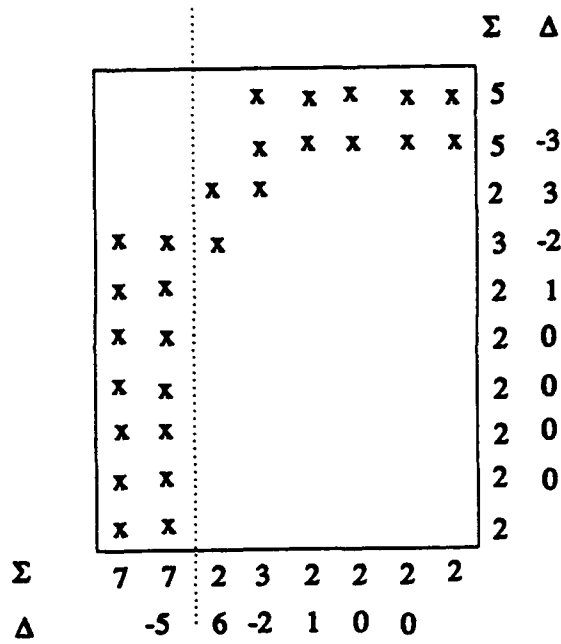


Figure 2 shows the signature arrays Σ_x, Σ_y and the second derivatives Δ_x, Δ_y used to partition the clusters.

be written in flux form, i.e.,

$$U_{ijk}^{n+1} = U_{ijk}^n - \Delta t \left[\frac{F_{i+n,j,k} - F_{i-n,j,k}}{\Delta x} + \frac{G_{i,j+n,k} - G_{i,j-n,k}}{\Delta y} + \frac{H_{i,j,k+n} - H_{i,j,k-n}}{\Delta z} \right] \quad (3.1)$$

where F, G, H are the numerical fluxes in the x, y, z directions respectively. In its current form, these numerical fluxes are assumed to be explicitly computable from the values in cell ijk and a localized collection of its neighbors, as is typical of conventional explicit finite difference methods. When the integrator is invoked, it is provided data on the grid to be integrated as well as sufficient boundary data (based on the scheme's stencil) to advance the solution on the given grid. No special stencils are used at fine/coarse interfaces. Instead, coarse grid data is linearly interpolated to the fine grid resolution to provide a border of boundary cells, which is provided to the integrator along with the grid data itself. After the integration step, fluxes are adjusted to insure conservation at grid interfaces.

When operator splitting is used with local grid patches, the only thing to note is that extra boundary cells must be integrated during the first sweep to provide accurate boundary values for subsequent sweeps. For example, for a scheme with four points to the side in the stencil, 4 entire rows of dummy cells along the top and bottom of the grid must be advanced in the x sweep, so that 4 points are available for the y sweep at the next stage. For very small grids this additional boundary work can dominate the computational cost of advancing a grid (particularly for difference methods having a broad stencil such as the Godunov algorithm we are using). This additional boundary work, as well as the vectorization issues, place a premium on generating large but efficient grids during regridding.

As an example, if one $60 \times 60 \times 60$ grid were replaced by two $30 \times 60 \times 60$ grids, both grids would redundantly integrate the overlapping boundary cells. This causes approximately 4% of the total computation to be redundant.

4. Implementation

Two simple decisions had a large impact on the implementation of AMR. The first concerned the organization and separation of the problem dependent and problem independent parts of the AMR code. The problem dependent parts include the particular hyperbolic systems to be solved (and a suitable integration scheme), the initial and boundary conditions, the problem domain, and the error estimation routines. When a new problem is being set up the changes required to the code are localized to a small number of subroutines. The integration subroutine advances the solution of the particular differential equations for a single timestep on a rectangular subgrid, and returns fluxes that are required to insure conservation at coarse-fine boundaries. Consequently, adapting an existing integration module for use with the AMR algorithm is routine. The remainder of the AMR shell treats the data in terms of conserved variables where the number of variables is specified as a parameter. Thus, the data structures, memory management, grid generation algorithms, the logic controlling the timestepping and subcycling on subgrids, interior boundary conditions, and the interfacing between grids that insure conservation are completely divorced from the

particular system being solved.

The second implementation detail that simplified the programming of AMR and also resulted in much cleaner code than in [2] is the use of a global integer index space covering the domain. These integers are used in describing the location of the grids. Based on the initial (user-specified) domain, given in floating point numbers, an integer lattice based on the number of cells in each dimension (n_x, n_y and n_z) is determined. The domain may then be partitioned into several coarse grids, each located in subcubes with indices between (1,1,1) and (n_x, n_y, n_z). If the refinement ratio is r between level l and level $l+1$, then the fine grid cells corresponding to coarse grid cell i, j, k are $r \cdot i + p_i, r \cdot j + p_j, r \cdot k + p_k$ where the $p_{i,j,k}$ independently range from 0 to $r-1$. This completely eliminates round-off error problems that would otherwise require careful coding to determine whether two grids overlap or whether a coarse grid is a parent to a fine grid.

We are currently rewriting the AMR algorithm in C++ with calls to FORTRAN routines for the numerically intensive parts. By constructing the appropriate classes we are able to define a grid calculus in which the computation of intersections and, in fact, the entire regridding process, is greatly simplified. Using the built in macro preprocessor, we are able to implement a large portion of the code in a dimension independent manner with the dimension as a compile time parameter. With the data hiding inherent in C++ we are able to implement AMR in such a way that the underlying data representation is restricted to a few model dependent classes. Changing the data representation, such as to a sparse data representation for a multi-fluid version of the code, would be restricted to the member functions of these classes. Since the majority of the run time for AMR is spent integrating large rectangular meshes, the overhead experienced by doing the data management in C++ is just a few percent over an implementation done entirely in FORTRAN. Further optimization of the most important member functions reduces the running time to a few percent less than the pure FORTRAN code.

5. Numerical Example

To test the performance of the 3DAMR algorithm, we have modeled the interaction of a Mach 1.25 shock in air hitting an ellipsoidal bubble of Freon-22. The case being modeled is analogous to one of the experiments described by Haas and Sturtevant [7]. The Freon is a factor of 3.0246 more dense than the surrounding air which leads to a deceleration of the shock as it enters the cloud and a subsequent generation of vorticity that dramatically deforms the bubble.

The computational domain is a rectangular region with length (x) 22.5 cm and width (y) and height (z) of 8.9 cm. The major axis of the bubble is 3.5 cm and is aligned with the z -axis. The minor axes are 2.5 cm with circular cross-sections of the bubble in the x - y plane. The bubble is centered at the point $(x, y, z) = (10 \text{ cm}, 0 \text{ cm}, 0 \text{ cm})$. The shocked fluid is placed at points less than or equal to 14.5 cm in the x direction. The shock moves in the direction of increasing x . We use the operator split second order Godunov method of [9], with Strang splitting. Reflecting boundary conditions are set on the constant z and

constant y planes. The inflow and outflow velocities on the constant x planes, as well as the interior fluid velocities, are set so the frame of reference is shifted to one in which the post-shock velocity is zero to minimize the x extent of the problem. The numerics preserve a four-fold symmetry in y and z , so we only compute on a quarter of the physical domain. (We have reflected the data in the renderings so that the entire domain is shown.)

We use a simplified treatment of the equations of multifluid gas dynamics. The features of this method include using a single fluid solver, only having to advect one additional quantity, and having a set of equations in conservation form.

We use a γ -law equation of state for each gas with $\gamma_a = 1.4$ for air and $\gamma_f = 1.25$ for Freon. Mixtures of the two gases are modeled with an equation of state defined using both γ 's,

$$\Gamma_c = \frac{1}{\frac{f}{\gamma_f} + \frac{(1-f)}{\gamma_a}}$$

for sound speeds and

$$\Gamma_e = 1 + \frac{1}{\frac{f}{\gamma_f - 1} + \frac{(1-f)}{\gamma_a - 1}}$$

for energy. Here, Γ_c is used to compute an effective sound speed for the mixture with

$$c = \sqrt{\frac{\Gamma_c p}{\rho}} \quad , \quad (5.1)$$

and f is the mass fraction of the freon. The harmonic average used to compute Γ_c , used by Colella, Ferguson and Glaz for multifluid computations [10], expresses the net volume change of a mixture of the gases in terms of their individual compressibilities. The sound speed defined by (5.1) is used in the integration routine for defining characteristic speeds and for approximate solution of the Riemann problem solution. We also assume that the two components of a mixed fluid cell all are at the same pressure. Pressure is computed from density and internal energy using Γ_e , namely,

$$p = (\Gamma_e - 1) \rho e \quad .$$

The harmonic average used to compute Γ_e insures that mixing of the two fluids at the same pressure does not result in a pressure and internal energy change of the composite fluid.

The AMR algorithm was run with an initial coarse grid of $80 \times 16 \times 16$ with two levels of refinement, each by a factor of 4, for 100 coarse grid time steps. The integration used 6 conserved quantities (mass, momentum, energy, and mass of Freon). The addition of this extra conserved variable besides the usual 5 found in 3-dimensional gas dynamics doesn't change the AMR structure). The error estimation procedure was modified so that the finest level grids (level 3) only existed in a neighborhood of the bubble and so that acoustic waves away from the bubble were not refined once the incident shock was well past the bubble. The computation was performed on a Cray-2 and required 20.16 hours of CPU time. In Figure 3, we show

volume renderings of the density field at four times during the evolution of the cloud. For the renderings, we interpolated all the data on to a uniformly fine grid. The effective result of the volume rendering is to yield an isosurface of the interface between air and freon. The earliest frame is shortly after the incident shock has completely passed through the bubble. The bubble evolution qualitatively agrees with the experimental results of Haas and Sturtevant. The dominant features of the flow are a strong jet coming from the back of the bubble and the unstable evolution of the roll-up of the outer portion of the bubble.

During the computation a total of 1.77×10^9 cells were advanced by the integration routine (not including boundary advancements required by operator splitting) for an effective time of 40 μ -seconds per zone including all of the AMR overhead. Figures 4 and 5 present a more detailed breakdown of the algorithm performance. Figure 4 is a histogram of each linear dimension of the level 3 grids, an appropriate measure of the quality of the grids for an operator-split integration algorithm. For this computation the maximum grid dimension was limited to 50 and most grids were at least 24 cells wide in each direction. This was done to limit an individual grid size to 125000 zones times 6 variables per zone. Figure 5 shows the number of level 3 cells as a function of time and indicates a growth in memory requirements as the solution complexity grows. The maximum memory required at any point during the run was 22.6 Mwords.

Independent measurements of the integration algorithm on a single large grid $160 \times 64 \times 64$ indicated a time of 30 μ -seconds per zone. Thus, the additional boundary work, smaller vector lengths and AMR overhead increased the time per zone by 10 μ -seconds. However, to achieve the same resolution, a uniform grid $1280 \times 256 \times 256$ would be required. Such a computation would require 500 Mwords of storage and 1100 hours of CPU time. The net speedup with AMR is a factor of 55.

Although it is difficult to predict in advance, only 40 of the 80 zones in the x direction played a significant role in the computation. The difficulty in prediction of the extent of the problem is shown by cruder mesh calculations which indicated a larger computation region. An outflow boundary condition to handle the reflected shock off of the bubble could have eliminated a region of length 40 zones in the x direction. Such a computation would require 251 Mwords of storage and 560 hours of CPU time. Alternatively, we estimate that with a carefully designed, exponentially stretched computation mesh, the fixed grid computational cost also could be halved; however, such an approach also requires substantial knowledge of the solution in designing the mesh. Thus, even from a conservative viewpoint, AMR reduced the computational cost by more than a factor of 20 for this problem.

Of course, the performance of AMR is highly problem dependent. For some problems the cost reduction may be greater and for some problems it may be less. However, AMR will be cost effective as long as the average number of coarse cells per time step that require the finest level of refinement over the entire course of the computation is less than 75% of the total number. We also note that sophisticated grid placement strategies can reduce the advantage of using AMR. However, these strategies require considerable knowledge of the solution to be effective and may add considerable difficulties to problem setup. AMR provides a high level of performance while making problem setup routine.

6. Acknowledgements

The work of J. Bell and M. Welcome was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. Partial support under contract No. W-7405-Eng-48 was provided by the Applied Mathematical Sciences Program of the Office of Energy Research and by DNA under IACRO 90-871. J. Saltzman was supported by the U.S. Department of Energy through Los Alamos National Laboratory under contract No. W-7405-ENG-36. M. Berger was supported in part by AFOSR Grant 86-0148, DOE Grant DE-FG02-88ER25053, and by a PYI (NSF Grant ASC-8848101)

References

1. M. J. Berger and J. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," *J. Comp. Phys.*, vol. 53, pp. 484-512, 1984.
2. M. J. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *J. Comp. Phys.*, vol. 82, pp. 64-84, 1989.
3. P. Colella and L. F. Henderson, "The von Neumann paradox for the diffraction of weak shock waves," *J. Fluid Mech.*, vol. 213, pp. 71-94, 1990.
4. P. Colella, R. Klein, and C. McKee, "The Nonlinear Interaction of Supernova Shocks with Galactic Molecular Clouds," in *Proceeding of the Physics of Compressible Turbulent Mixing International Workshop*, Princeton, NJ, 1989.
5. L. F. Henderson, P. Colella, and E. G. Puckett, "On the Refraction of Shock Waves at a Slow-Fast Gas Interface," *J. Fluid Mech.*, to appear.
6. I. I. Glass, J. Kaca, D. L. Zhang, H. M. Glaz, J. B. Bell, J. A. Trangenstein, and P. Collins, "Diffraction of planar shock waves over half-diamond and semicircular cylinders: an experimental and numerical comparison," in *Proceeding of the 17th International Symposium on Shock Waves and Shock Tubes*, Bethlehem, PA, July 17-21, 1989.
7. J.-F. Haas and B. Sturtevant, "Interaction of weak shock waves with cylindrical and spherical gas inhomogeneities," *J. Fluid Mech.*, vol. 181, pp. 41-76, 1987.
8. M. J. Berger and I. Rigoutsos, "An Algorithm for Point Clustering and Grid Generation," NYU Technical Report #501, NYU-CIMS, to appear in *IEEE Trans. Systems, Man and Cybernetics*.
9. P. Colella and H. M. Glaz, "Efficient Solution Algorithms for the Riemann Problem for Real Gases," *J. Comput. Phys.*, vol. 59, pp. 264-289, 1985.
10. P. Colella, R. E. Ferguson, and H. M. Glaz, *Multifluid Algorithm*, to appear.

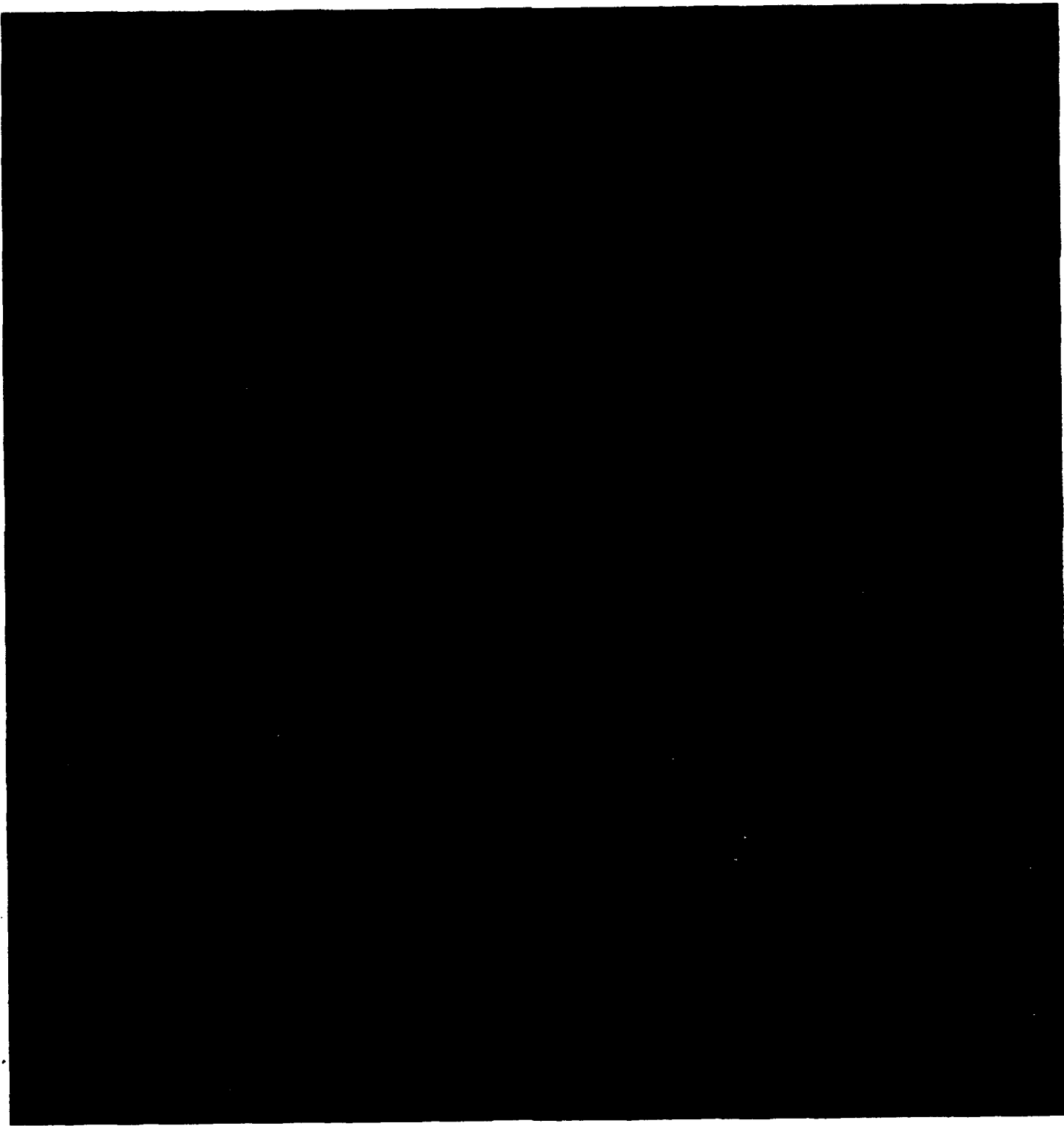


Figure 3 Volume renderings of the density during the evolution of the bubble after being hit with a shock wave.

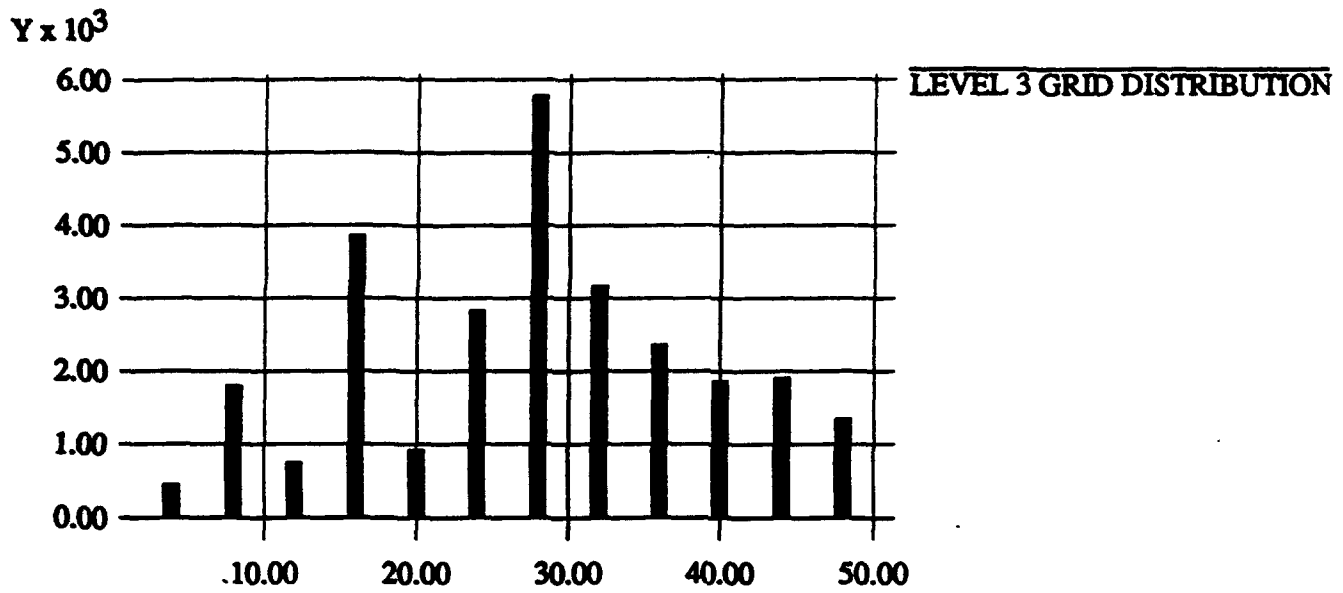


Figure 4 histograms the linear dimensions of all the level 3 grids.

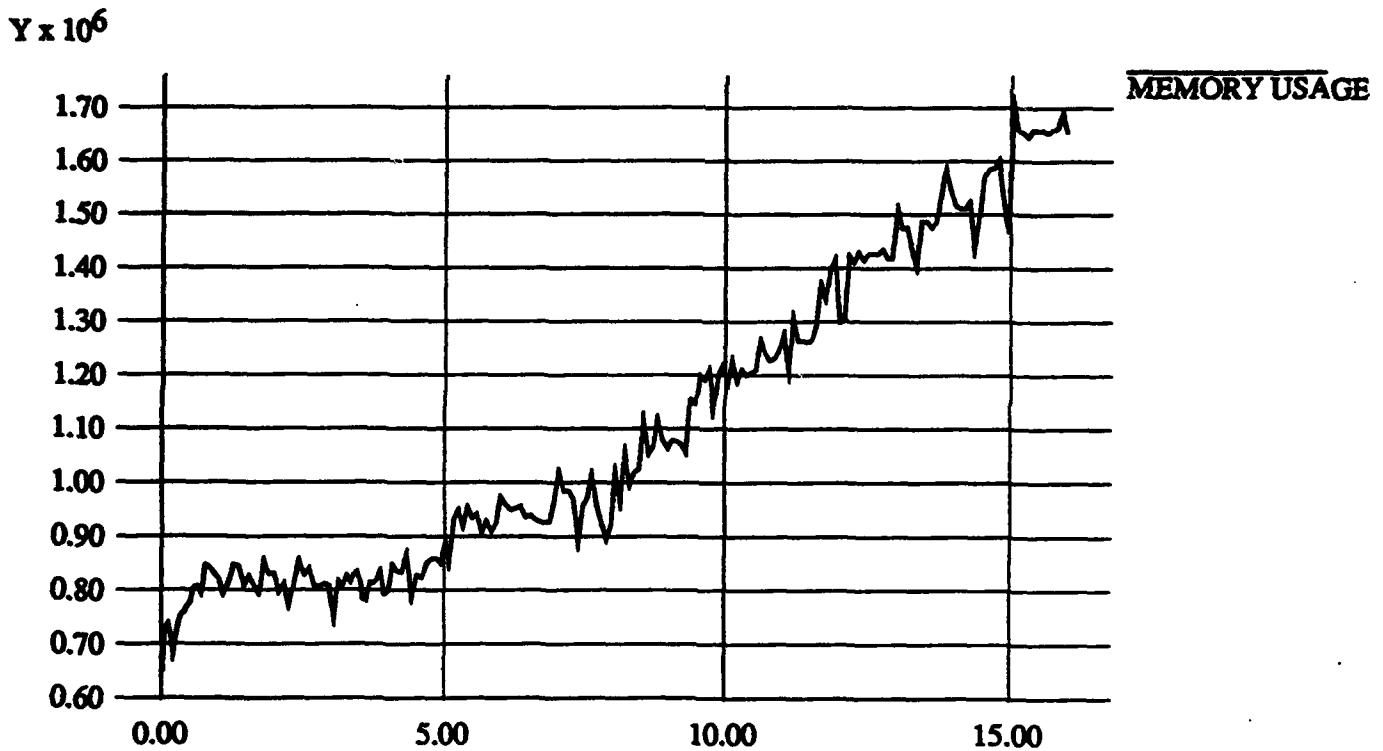


Figure 5 shows the number of level 3 cells as a function of time during the calculation.

A Rotated Difference Scheme for Cartesian Grids in Complex Geometries*

Marsha J. Berger

Courant Institute of Mathematical Sciences
New York University
New York, NY 10012

Randall J. LeVeque

Mathematics Departments
University of Washington, Seattle, WA 98195
and ETH, Zurich, Switzerland

Abstract

We present a Cartesian mesh algorithm with adaptive refinement to compute transient flows with strong shocks around arbitrary geometries. We develop a rotated difference scheme for use at the arbitrarily small cells where the Cartesian grid intersects the body. Our scheme is stable using a time step based on the regular cells away from the body. Adaptive mesh refinement is used to achieve high resolution in the solution. We propose a simple but useful test problem with a smooth solution for comparing schemes for arbitrary geometries such as ours.

1. Introduction

We present a method for computing dynamic compressible flows with strong shocks about complex geometries. The method combines three ingredients: a high resolution shock capturing scheme of Godunov type, an adaptive mesh refinement strategy, and a new Cartesian mesh method for using rectangular meshes about arbitrary geometries. Since the first two ingredients have been well described elsewhere [9,1], we concentrate on new problems that arise in Cartesian mesh calculations about general geometries. Some preliminary numerical results confirm the usefulness of the method.

The overall goal of this work is to preserve the advantages of uniform grid methods even for flows about complex geometries. These advantages include simplicity and efficiency of data structures, and greater accu-

racy and resolution of the solution, particularly for complex flows with strong interacting shocks. High local resolution can be achieved where needed through local uniform fine grids patches, which can be recursively nested for accuracy. This technology has been fully demonstrated before [2,1,5]. Although the boundary scheme discussed below is somewhat more complex than the interior scheme, this extra work is done only at boundaries and it does not contribute significantly to the overall computational cost of a computation.

We use a finite volume method to solve the system of conservation laws

$$u_t + f(u)_x + g(u)_y = 0.$$

Let $u_{i,j}^n$ be the approximate solution in cell (i,j) at timestep n . Then in each time step we update the cell values using the flux differencing formula

$$u_{i,j}^{n+1} = u_{i,j}^n - \frac{\Delta t}{A_{i,j}} \sum F \cdot l, \quad (1)$$

where the sum is over the faces bounding the (i,j) cell, $A_{i,j}$ is the cell area, l is the length of the interface, and F is the flux in the normal direction. Figure 1 shows a typical Cartesian grid and the irregular cells near the solid wall boundary. Adjacent to the boundary, a cell can have 3, 4 or 5 edges. Also note that cell areas can be orders of magnitude smaller than the area h^2 of the regular cells, as is the area of cell (2,1) in Figure 1. Equation (1) is used to update all cells; however, the flux is computed in a special way at the irregular boundary cells.

There are two main tasks in our treatment of the small irregular cells that arise at the boundary where a solid object intersects the Cartesian mesh. The first is to

*AIAA Paper CP-91-1602.

develop a stable difference scheme for these boundary cells for use in conjunction with a time dependent finite volume scheme. The time step should be based on the area of the regular cells away from the boundary; it must not be constrained by tiny cells near the boundary.

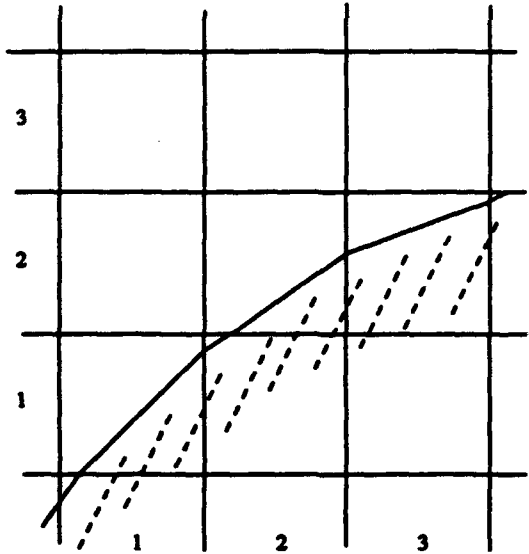


Figure 1 A typical Cartesian mesh. Cell (2,1) is tiny. Cell (2,2) has 5 edges.

The second task is to develop an approach that maintains accuracy in regions of smooth flow, even on irregular grids. This appears to be much more difficult to achieve and we can report here only partial success. We have improved the method proposed in [3,4] by introducing a piecewise linear representation near the boundary in an attempt to compute second order accurate fluxes. Typical second order accurate methods rely on cancellation of errors in the fluxes during the flux differencing. This occurs only for smoothly varying grids. Our irregular boundary cells are not at all smooth. However, recent results on "supraconvergence" [16] indicate that with appropriate methods the local error can be one order lower than normally required in irregular grid cells without the error accumulating to destroy the global accuracy. We are currently attempting to perform such analysis for methods of the type proposed here. The flux redistribution method of Chern and Colella [6] also stabilizes calculations with small cells but apparently at an even greater cost in accuracy.

Careful investigation of the order of accuracy is often neglected for CFD methods, particularly for complex geometries where challenging test problems with known solutions seem to be rare. Here we propose a simple test problem of expanding flow through a curved channel for which the exact steady state solution can be computed using characteristic theory. Our current method achieves only a first order convergence rate at the boundary, and a second order convergence rate in the interior. Since our solid wall boundary condition is still only first order accurate, this is the most we could expect.

The difference scheme we have developed uses rotated coordinates at each boundary cell, in directions normal and tangential to the boundary. A similar difference scheme, where the directions were chosen to be aligned with the flow field, was developed in [13]. We describe our work in the context of the MUSCL scheme we use to update the cell in the interior of the flow field. However, these ideas extend naturally to other difference schemes, and some of the results we show in the last section were obtained using an extension of central differencing to this rotated framework, with a Runge Kutta method for the time stepping.

Earlier work using Cartesian meshes in the 1980's also used a central differencing method with Runge Kutta in time [8,10,14], as well as flux vector splitting [7]. A newer approach [17], (presented at this conference), also combines a MUSCL type scheme with Runge Kutta time stepping. This new approach incorporates adaptive grid refinement as well. All of these approaches are for steady state computations. Time accurate simulations, which we are focusing on, are particularly difficult because of the stability issue for the small cells, and the difficulty in obtaining second order accuracy in time as well as space. Unsteady shock calculations for irregular grids are also more difficult than their steady counterparts [13]. We have previously investigated several ways in which stability can be achieved in the presence of small cells [4,11,12]. The point of this work is to improve the accuracy of the boundary scheme.

2. The Rotated Difference Scheme

We describe the implementation of the rotated difference scheme in the context of a MUSCL scheme. Great simplifications are possible however if it is imple-

mented with a central difference scheme instead. We start with a very brief review of the MUSCL method, using the approach of [9]. To compute a flux at the interface $(i+1/2, j)$, for example, left and right states u_L, u_R at the midpoint of the interface and at time $n+1/2$ are predicted. A solution to the Riemann problem with this input data produces the flux $f_{i+1/2, j}^{n+1/2}$ that is used to update the solution in eq. (1). Since the flux is centered at time $n+1/2$, it is second order accurate in time. Roughly speaking, a linear reconstruction of the solution, using limited slopes to prevent overshoots, computes the solution to second order accuracy in space at the cell edges.

Looking in more detail at the Taylor series for the approximation of u_L gives

$$\begin{aligned} u_L^{n+1/2} &= u_{i,j}^n + \frac{\Delta x}{2} u_x + \frac{\Delta t}{2} u_t \\ &= u_{i,j}^n + \frac{\Delta x}{2} u_x - \frac{\Delta t}{2} (f_x + g_y) \\ &= u_{i,j}^n + \frac{\Delta x}{2} \left(1 - \frac{\Delta t}{\Delta x} \frac{\partial f}{\partial u} \right) u_x - \frac{\Delta t}{2} g_y \end{aligned} \quad (2)$$

In the MUSCL scheme we are employing, the slopes u_x and u_y are approximated using finite differences and slope limiting in the primitive variables. We have extended this to also obtain estimates of the gradient in irregular cells near the boundary. The term g_y is called the transverse derivative, and is the most difficult term to compute in our rotated difference scheme.

If the derivatives are computed with first order accuracy, the resulting approximation at the interface is second order accurate. For smooth grids, the leading order error terms cancel when fluxes are differenced, giving a local truncation error of order $(\Delta t \cdot \Delta x^2)$. As indicated above, the situation with irregular grids is less clear.

Near the boundary, we replace the MUSCL scheme by a rotated version, in which the flux across each interface is computed as a linear combination of a flux f_{normal} in the direction (α, β) normal to the boundary and a flux f_{tan} in the orthogonal direction $(\beta, -\alpha)$ tangential to the boundary. Here (α, β) is the unit normal to the boundary in cell (i, j) as shown in Figure 2. The full flux is then given by

$$f_{i+1/2, j} = \beta \cdot f_{tan} - \alpha \cdot f_{normal}$$

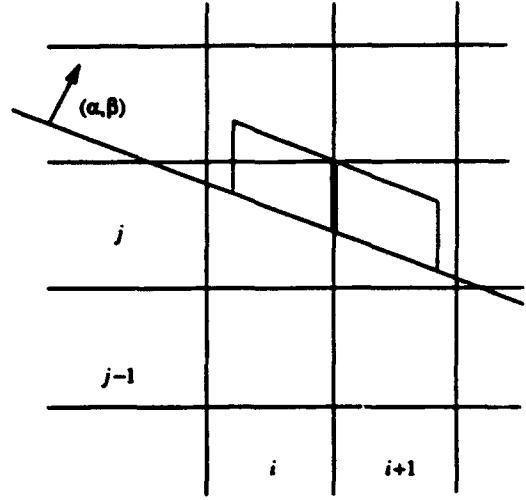


Figure 2 Tangential boxes constructed for a vertical interface.

The fluxes f_{normal} and f_{tan} are computed based on appropriate data u_L and u_R . For example, in Figure 2 the tangential flux at the vertical interface $(x_{i+1/2}, y_j)$ would be computed based on data that approximates the average value of u in the boxes extending a distance h from the interface in the directions tangential to the boundary. A cell centered approximation to the solution in these boxes is computed by area weighted averaging of the solution on the underlying Cartesian mesh. Our previous work [4,5] used area weighted averaging of the piecewise constant function given by the cell averages u_{ij}^n . A second order accurate approximation is computed by using a linear reconstruction of the solution on the Cartesian grid instead, based on the slopes u_x and u_y obtained previously via slope limiting. For central differencing, nothing more would be needed. For the MUSCL scheme, we also compute slopes for each box using area weighted averaging of the slopes in the underlying Cartesian grid. This gives a first order accurate approximation to the gradient in the box. In two dimensions, the tangential boxes intersect at most two grid cells. The weighting coefficients can be precomputed for each boundary cell and used for the entire calculation.

The motivation for using a rotated difference scheme and choosing the data based on area weighted averaging is described in [4], where it is shown that this approach leads to fluxes which cancel out to $O(\Delta_{ij})$ when we compute the flux differences in (1). This gives

stability even when the cell has a very small area A_{ij} relative to the time step Δt . Note that for tiny cells, this has the effect of increasing the stencil of the scheme, so that the numerical domain of dependence extends at least a distance h in all directions and the CFL condition remains satisfied.

Given the cell centered states in the tangential boxes, left and right states at the interface are computed using eq. (2) appropriately rotated into the tangential framework. The only missing piece is the transverse derivative. A discussion of this is deferred until after the solid wall boundary condition is presented.

This same procedure is also applied in the direction normal to the boundary at each cell edge. However, the normal boxes may extend outside the flow field, into the solid body itself (see figure 3). In this case we need values from outside the flowfield to use in the area weighted average of states, and we use the values from the cells *outbox* which is described in the next section. In the present implementation, we do not use slopes in computing left and right states at cell edges for the normal Riemann problem.

We have been careful in designing the components of the algorithm, such as the slope limiting and transverse derivatives, to satisfy the following design criterion. If we compute flow through a straight channel aligned with the grid, and the data is one-dimensional (varying only with length along the channel), then the solution should remain one-dimensional and not suffer two-dimensional distortion due to the small cells at the boundary. This requires that the fluxes for the irregular cells be consistent with the fluxes computed by the MUSCL scheme at the regular interfaces.

3. Solid Wall Boundary Conditions

At the solid wall boundary itself the flux can be computed more simply, by solving a single Riemann problem normal to the boundary. This is shown in figure 3. For each boundary segment we create a box of length h in the direction (α, β) . This is called the *inbox* for cell k . An approximate solution q_k in this box is again obtained using area weighted averages of the linearly reconstructed solution on the Cartesian mesh. The solution is rotated into the boundary coordinate frame and we then define \bar{q}_k , the value in the corresponding *outbox*, by negating the normal component of q_k . A boundary

Riemann problem is solved between q_k and \bar{q}_k to satisfy the boundary conditions of no normal flow. The value \bar{q}_k in the outbox of cell k is also used to help reconstruct the solution for the normal Riemann problem at interior cell edges, as described in the previous section. Since no slopes are used along with the inbox and outbox solution approximation, this boundary condition is only first order accurate.

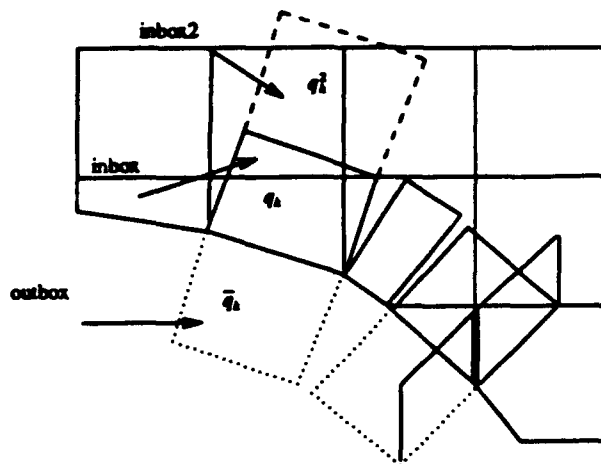


Figure 3 The boundary flux is computed by a boundary Riemann problem in the normal direction. The outbox contributes to the normal Riemann problem for interior edges as well.

To obtain the transverse derivative needed in the MUSCL scheme for tangential Riemann problems, we need to estimate flux derivatives normal to the wall (corresponding to the g_y term in (2)). Two rows of inboxes are created normal to the boundary, each of length h in the normal direction. The approximate solution in the second inbox, q_k^2 , is again computed using area weighted averaging of the piecewise linear function defined on the Cartesian grid. Riemann problems between q_k and q_k^2 give us one flux, call it g_k^2 . The boundary Riemann problem gives us another flux \bar{g}_k . The flux difference $(g_k^2 - \bar{g}_k)/h$ approximates the normal derivative $\partial g / \partial n$ to first order.

In solving Riemann problems normal to the boundary, we also need to define transverse derivatives. These are computed by solving Riemann problems based on data in adjacent inboxes along the boundary. However, because the width of these inboxes varies and may

be small relative to h , we actually combine the values from several inboxes, based again on area weighting averaging, to define values over boxes of width h .

4. Computational Examples

We present two computational examples illustrating the use of this method. In the first problem an incident shock reflects off two cylinders that are offset to each other. A complicated pattern of reflection is set up between the cylinders. The second test case, described in Whitham [16], has a smooth steady state solution. Although we are not primarily interested in steady state calculations, we use this problem to compute the order of accuracy of the method, and propose it in general as a good test problem for irregular grids. The code is written to handle general geometries. The only difference between these two examples is the description of the geometry of the solid bodies.

4.1 Flow past two cylinders

In the first example we study the behavior of an incident shock traveling at Mach 2.31 and its reflection off of two cylinders. The initial conditions are

$$\begin{pmatrix} \rho_L \\ u_L \\ v_L \\ p_L \end{pmatrix} = \begin{pmatrix} 5.1432 \\ 2.04511 \\ 0.0 \\ 9.04545 \end{pmatrix} \quad \begin{pmatrix} \rho_R \\ u_R \\ v_R \\ p_R \end{pmatrix} = \begin{pmatrix} 1.4 \\ 0.0 \\ 0.0 \\ 1.0 \end{pmatrix}. \quad (3)$$

The computational domain is the unit square, and the initial shock location is at $x=.27$. We use a 50 by 50 coarse grid, with one level of adaptive mesh refinement by a factor of 4. Figure 4 shows contour plots of the solution at several times, along with body plots of the pressure around each cylinder at selected times. The body plots start at the back of each cylinder and travel clockwise around the object. In the body plots, object one is the top cylinder, and object two is the lower cylinder. The contour plots also include an outline of the location of the refined grids. The approximate solution is taken from the finest grid in the region. An indication of the grid spacing can be seen from the outline of the irregular cells at the solid wall boundary.

Despite the complete irregularity of the boundary cells, the pressure plots around the cylinders are quite smooth. By time .3, the solution has become quite com-

plex. The pressure plot shows the two shocks incident on the lower cylinder, just above the pressure peak at the leading edge. A third level of refinement could be used to show more fine structure here. The incoming pressure ratio for this problem is 9 to 1. The pressure ratio across the reflected shock is roughly 40 to 1.

4.2 Smooth test problem

In order to study the accuracy of our boundary treatment, we consider a steady state problem consisting of smooth flow in a curved channel. We choose the channel in such a way that the exact solution consists of a simple rarefaction wave and can be approximated to arbitrary accuracy using characteristic theory. We will briefly describe this procedure here. More details can be found in Section 6.17 of Whitham [16].

The lower wall of the channel is given by the function

$$y_w(x) = \begin{cases} 0.302 & x \leq 0.1 \\ 0.302 - 0.3(x-0.1)^2 & 0.1 \leq x \leq 0.7 \\ 0.194 - 0.36(x-0.7) & 0.7 \leq x \leq 2.24 \end{cases}$$

At the inflow boundary we use a Mach 1.30 supersonic flow given by

$$\begin{pmatrix} \rho \\ u \\ v \\ p \end{pmatrix} = \begin{pmatrix} 21.4856 \\ 1.0 \\ 0.0 \\ 9.04545 \end{pmatrix}.$$

In the absence of an upper wall, the resulting steady state solution is a simple wave in which the flow variables are constant along characteristics, which are straight lines. Let $\theta(x) = \tan^{-1}(y_w'(x))$ be the angle of the wall to the horizontal at each point x . Then the characteristic originating at the point $(x, y_w(x))$ makes an angle $\theta(x) + \mu(x)$ with the x -axis, where μ is computed from θ by solving the nonlinear equation

$$P(\mu) - P(\mu_0) = \theta$$

where

$$P(\mu) = \delta \tan^{-1}(\delta \tan \mu) - \mu$$

with

$$\delta = \sqrt{(\gamma+1)/(\gamma-1)}.$$

Here μ_0 is the characteristic angle in the free stream,

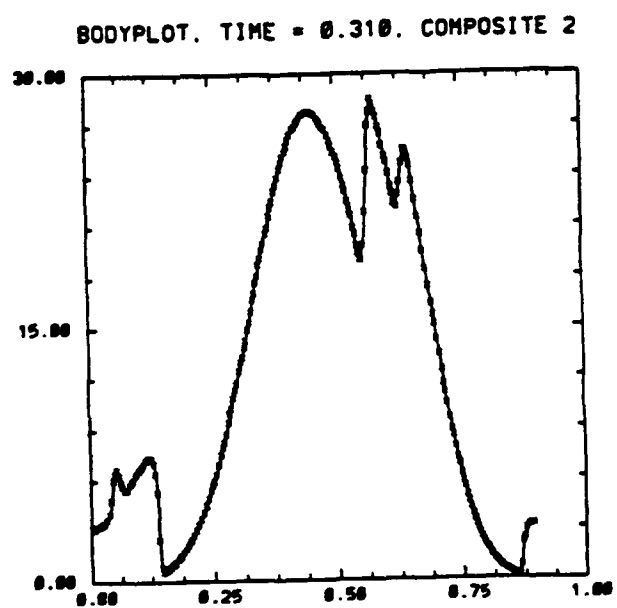
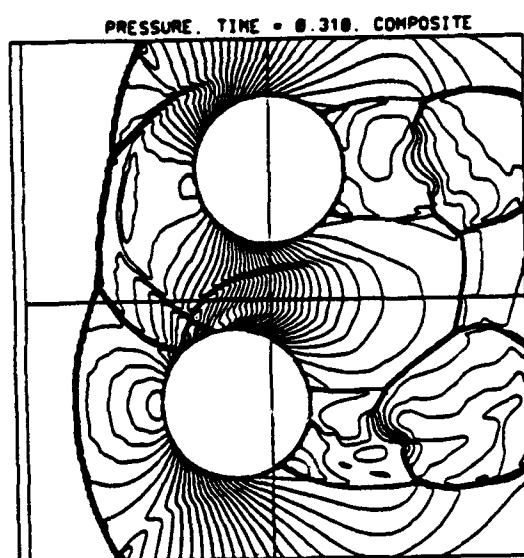
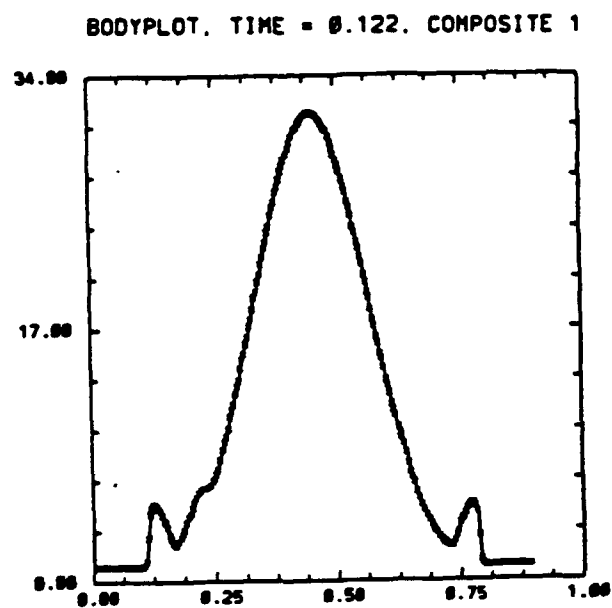
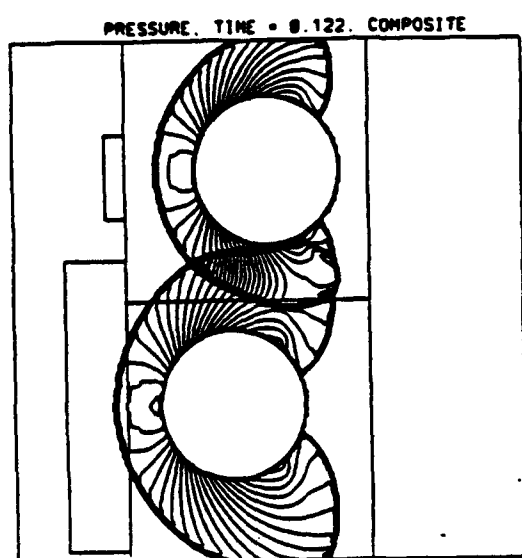
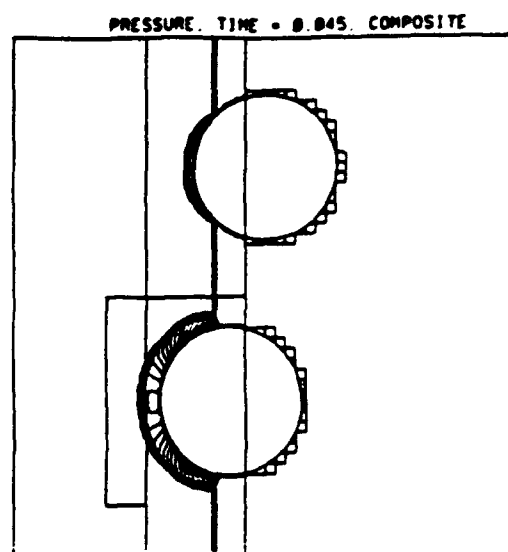
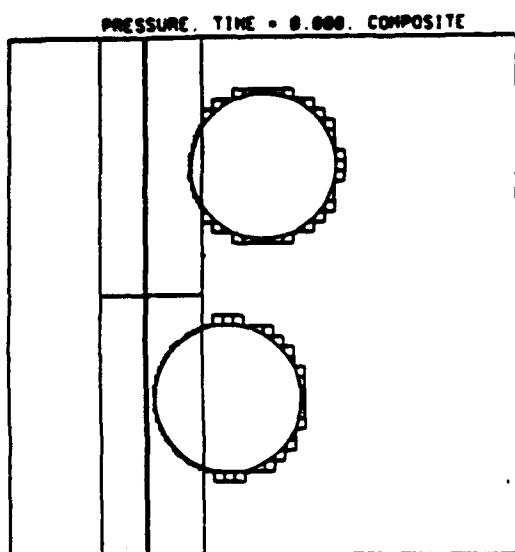


Figure 4 shows pressure plots of the flowfield and the pressure along the cylinders, at several different times. The location of embedded fine grids is indicated on the contour plots.

given by

$$\mu_0 = \sin^{-1}(1/M_0) \quad (M_0 = \text{Mach number}).$$

For a point $(x, y_w(x))$ lying on the wall, we can compute the value of the flow variables at this point and hence along the entire characteristic. We first compute the sound speed a in terms of the inflow sound speed $a_0 = \sqrt{\gamma p_0 / \rho_0}$ using

$$a = a_0 \left[\frac{1 + (\gamma - 1)/(2 \sin^2 \mu_0)}{1 + (\gamma - 1)/(2 \sin^2 \mu)} \right]^{1/2}.$$

The flow variables are then given by

$$u = a \sin \theta / \sin \mu$$

$$v = a \cos \theta / \sin \mu$$

$$p = p_0 (a/a_0)^{2\gamma/(\gamma-1)}$$

$$\rho = \gamma p / a^2$$

In our test problem, we define the upper wall to be a streamline of the flow, so that the exact solution is unaltered by the introduction of the wall. In practice we approximate this upper wall by a curve passing through a finite number of points. We first choose points (\bar{x}_j, \bar{y}_j) with $\bar{y}_j = y_w(\bar{x}_j)$ along the lower wall and then for each point determine a corresponding point (\hat{x}_j, \hat{y}_j) on the upper wall. The point (\hat{x}_j, \hat{y}_j) lies on the characteristic through the point (\bar{x}_j, \bar{y}_j) at a distance determined by the requirement that the mass flux across each such cross-section of the channel should be constant. After choosing an initial width w_0 for the channel at inflow, we can easily determine the corresponding distance at other points. In our test we have chosen $w_0 = 0.2$.

Finally, in order to evaluate the true solution at an arbitrary point (x, y) in the channel, we first find the point (\bar{x}, \bar{y}) along the lower wall that lies on the same characteristic, and then evaluate the flow variables at (\bar{x}, \bar{y}) as described above. To compute (\bar{x}, \bar{y}) from (x, y) , note that the slope of the characteristic must be $(\bar{y} - y)/(\bar{x} - x)$, leading to the nonlinear equation

$$(y_w(\bar{x}) - y) / (\bar{x} - x) = \tan^{-1}(\theta(\bar{x}) + \mu(\bar{x})),$$

which we can solve for \bar{x} using a numerical root finder. Figure 5 shows the exact solution in the interior of the flow field and along the walls.

We show the results of a convergence study in Table 1, using the MUSCL scheme with several different

grid sizes. We measure the relative error in density only along the boundaries, using the following L_1 norm,

$$\|e\|_1 = \sum_k \frac{\text{area}_k}{h} |e_k|$$

where the sum is only over the irregular boundary cells, divided by the same norm of the density. This weighted norm counts the larger irregular cells more heavily than the tiny ones, and scales correctly as the mesh is refined. (However, other L_1 norms give similar results).

h	boundary error (MUSCL)
.04	.78%
.02	.41%
.01	.20%

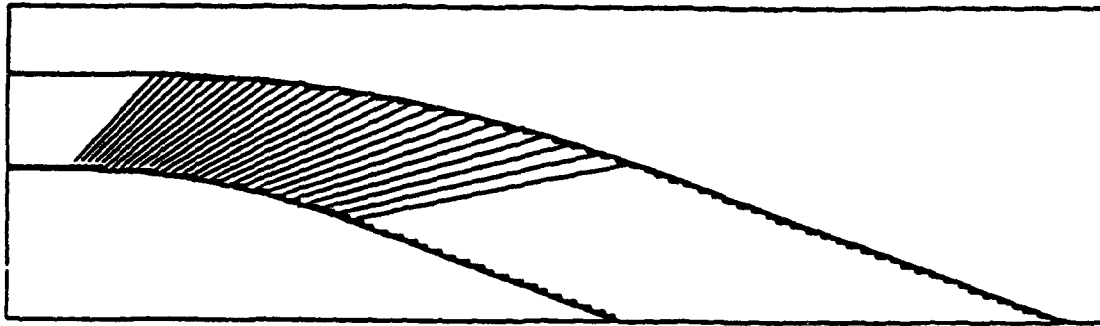
Table 1 The relative error in density in the converged solution in an L_1 norm along the boundary.

The boundary error is reduced by a factor of 2 when h is halved. The interior is converged to second order accuracy, despite the less accurate boundary scheme. The error in the interior is an order of magnitude smaller than the boundary error. We have also computed the local truncation error in the irregular cells, by taking a single time step of the method starting with the exact steady state as initial conditions. These results, shown in Table 2, also show a decrease by a factor of 2 as h is halved. Thus the global error behavior is similar to the local behavior, rather than losing a power of h as happens on uniform grids. For a comparison, our previous method of area weighted averages using a piecewise constant reconstruction of the solution gives approximately 30% larger errors at the boundary. Recent experiments using linear extrapolation at the solid wall boundary gives a factor of 2 improvement in the global error over the results in Table 1.

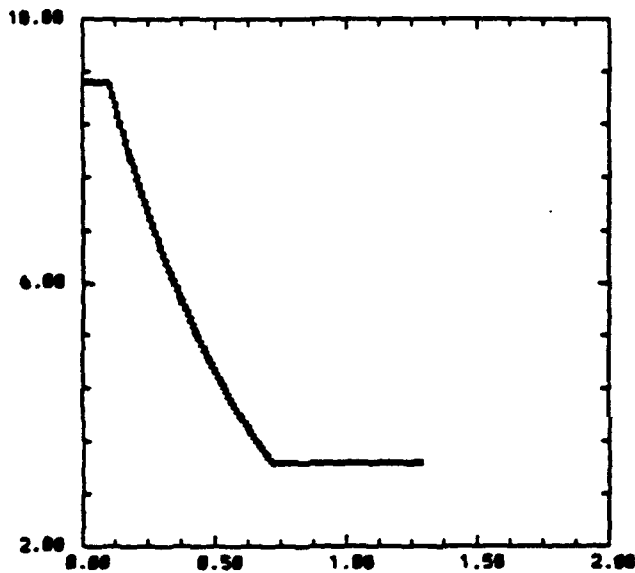
h	1 step error (MUSCL)
.04	.94(-1)%
.02	.48(-1)%
.01	.23(-1)%

Table 2 The relative local truncation error in an L_1 norm measured along the boundary for the MUSCL Scheme.

PRESSURE. TIME = 0.000. COMPOSITE



BODYPLOT. TIME = 0.000. COMPOSITE 1



BODYPLOT. TIME = 0.000. COMPOSITE 2

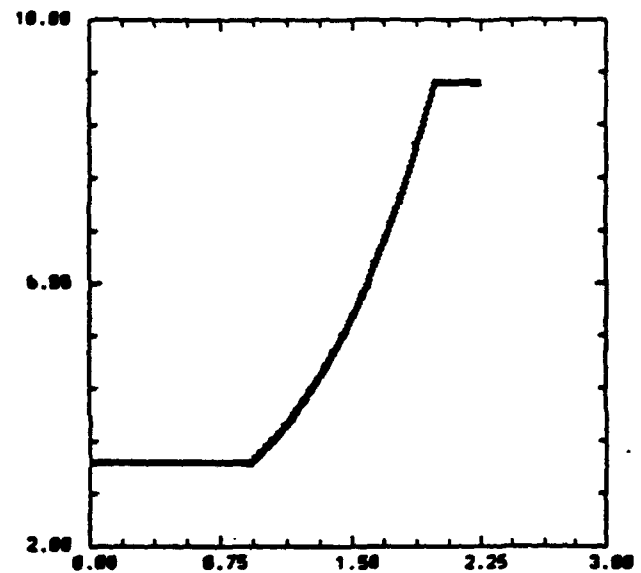


Figure 5 Pressure contours of the exact solution and the pressure along the walls.

We are continuing to investigate the behavior of the error for irregular grids. Despite this uncertainty, we feel this is a promising new way to simplify the grid generation problem for flows in complex geometries. Future work will focus on improving the overall accuracy, developing better limiters for computing the gradient at the irregular cells, and incorporating an improved solid wall boundary condition using the normal momentum equation to improve the prediction of the pressure in regions of high curvature.

Acknowledgments

We would like to thank John Bell, Phil Colella, Jonathan Goodman, and Jeff Saltzman for helpful discussions. This research was supported in part by the NSF PYI program (DMS-8657319 and ASC-8858101), by the AFOSR under Contract No. 91-0063, and by the Department of Energy under Contract No. DEFG02ER25053.

5. References

- [1] M. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics". To appear in J. Comp. Phys.
- [2] M. Berger and A. Jameson, "Automatic Adaptive Grid Refinement for the Euler Equations", AIAA J. 23,

(1985), pp. 561-568.

[3] M. Berger and R. LeVeque, "Cartesian Meshes and Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations". Proc. 3rd Intl. Conf. Hyperbolic Problems, Uppsala, Sweden, June, 1990.

[4] M. Berger and R. LeVeque, "Stable Boundary Conditions for Cartesian Grid Calculations", ICASE Report No. 90-37, May, 1990.

[5] M. Berger and R. LeVeque, "An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries", AIAA 89-1930, Buffalo, NY June, 1989.

[6] I. Chern and P. Colella, "A Conservative Front Tracking Method for Hyperbolic Conservation Laws". Submitted to J. Comp. Phys. UCRL-97200, July 1987.

[7] S. Choi and B. Grossman, "A Flux-Vector Split, Finite-Volume Method for Euler's Equations on Non-Mapped Grids", AIAA Paper 88-0227, Proc. 26th Aerospace Sciences Meeting, Reno, Nevada.

[8] D. Clarke, H. Hassan, and M. Salas, "Euler Calculations for Multielement Airfoils Using Cartesian Grids, AIAA Paper 85-0291, Jan. 1985.

[9] P. Colella, "Multidimensional Upwind Methods for Hyperbolic Conservation Laws", J. Comp. Phys. 87, 1990.

[10] R. Gaffney, H. Hassan and M. Salas, "Euler Calculations for Wings Using Cartesian Grids", AIAA Paper 87-0356, January 1987.

[11] R.J. LeVeque, "High Resolution Finite Volume Methods on Arbitrary Grids via Wave Propagation", J. Comp. Phys. 78 (1988), pp. 36-63.

[12] R.J. LeVeque, "Cartesian Grid Methods for Flow in Irregular Regions", in Numerical Methods for Fluid Dynamics III, K.W. Morton and M.J. Baines, editors, Clarendon Press (1988), pp. 375-382.

[13] D. Levy, K. Powell, and B. van Leer, "An Implementation of a Grid Independent Upwind Scheme for the Euler Equations", AIAA Paper 89-1931-CP, Buffalo, NY.

[14] J.J. Quirk, "An Adaptive Mesh Refinement Algorithm for Shock Hydrodynamics", Cranfield Institute of Technology CoA Report No. NFP89/07, March, 1989.

[15] B. Wedan and J. South, "A Method for Solving the Transonic Full-Potential Equations for General Configurations", Proc. AIAA Computational Fluid

Dynamics Conf., July 1983.

[16] B. Wendroff and A. White, Jr. "Some Supraconvergent Schemes for Hyperbolic Equations on Irregular Grids", Proc. 2nd Intl. Conf. Nonlinear Hyperbolic Problems, Aachen, 1988. (Notes on Numerical Fluid Mechanics, Vol. 24)

[17] G. Whitham, Linear and Nonlinear Waves, Wiley-Interscience, 1974.

[18] D. De Zeeuw and K. Powell, "An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations". These proceedings, AIAA-91-1542.

AMR on the CM-2

Marsha J. Berger and Jeff Saltzman

The Research Institute of Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 311, Columbia, MD 244, (301)730-2656

Work reported herein was supported in part by the NAS Systems Division of NASA via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035.

AMR on the CM-2

Marsha J. Berger
Courant Institute
251 Mercer Street
New York, NY 10012

Jeff S. Saltzman
Los Alamos National Laboratory
MS B265
Los Alamos, NM 87545

Abstract

We describe the development of a structured adaptive mesh algorithm (AMR) for the CM-2. We develop a data layout scheme that preserves locality even for communication between fine and coarse grids. On 8K of a 32K machine we achieve performance slightly less than 1 CPU of the Cray Y-MP. We apply our algorithm to an inviscid compressible flow problem.

1 Introduction

Local Adaptive Mesh Refinement (AMR) is an algorithm that can efficiently compute complex flow fields where only a small fraction of the computational domain needs to be resolved. Its success has been demonstrated over the last ten years in [1, 4, 3, 15, 20, 21]. A natural question is whether this approach is still viable on massively parallel architectures. In particular, can we still take advantage of local regularity of the grids? Can the dynamic and adaptive character of AMR be maintained while balancing the load on a fine grained data parallel/SIMD machine such as the CM-2? A new issue not found on serial architectures is minimizing inter-grid communication forced by the distributed memory.

In the last few years several adaptive unstructured mesh codes have been parallelized [7, 22, 17]. Unstructured mesh methods tend to have much different overheads and efficiencies than structured mesh codes. Their data structures are lists of elements/edges, leading to more words of storage per node than structured methods use. However, this generality and flexibility lends itself to a natural extension to both coarse grained and fine grained parallel architectures. The issues in the parallelization of unstructured methods are the creation of subdomain partitions, and the mapping of the partitions onto individual processor nodes to minimize global communication and balance the load. The communication costs seem to be the major source of inefficiency in these codes, even for non-adaptive unstructured parallel codes. For example, Barth and Hammond [14] report 50% of the run time on the CM-2

is spent in communications tasks. The free-Lagrange code X3D [7] has been measured to spend 93% of its time in communication [18].

In contrast, very little work on structured meshes has been done. Gropp and Keyes have developed interesting algorithms for adaptive elliptic equations on semi-structured meshes [13]. Similar looking meshes are found in the parallel AFAC algorithms of [19]. Dynamically adaptive parallel algorithms for hyperbolic equations on quad tree data structures have been developed in [5, 6]. This work uses a coarse grained model of parallel computation, with the parallelization coming from different ways of traversing the tree.

Most other work on parallelization of adaptive structured meshes, to our knowledge, has targeted AMR, and uses coarse-grained parallelization on small numbers of processors. Since the data structures in AMR keep track of entire grids, rather than individual grid points, a natural approach here is to distribute the grids to different processors. Berger [2] has done this on a shared memory Cray X-MP4/16, and Crutchfield [10, 11] on a 32 node BBN TC2000. Neither of these approaches can take advantage of massively parallel computers. Our largest 3D application so far has used on the order of 500 grids at a given time. Even allowing for future applications with several thousand grids, this coarse grained approach would not scale well for a machine with several thousand processors or more.

We have developed a data parallel implementation of a 2-D AMR code for the CM-2. (For a discussion of the CM-2 architecture see [16].) In this approach the individual points in a grid are distributed to processors. The key idea in our strategy is the data layout. We map the points of grids on different levels to minimize intergrid global communication and preserve locality. In addition, the serial algorithm was modified in several ways, in particular, we have restricted the adaptive grid patches to be a fixed size. We compare the efficiency of our implementation on an 8K CM-2 with a functionally equivalent implementation on a Cray Y-MP. We measure that the ratio of integration time to total CPU time of a typical run approaches 75% on the CM-2 while this ratio on the Cray Y-MP is closer to 85%. We also measure the grind time of our integration procedure, defined as the time to update one grid point one timestep. This is roughly equivalent to the Cray Y-MP time. Thus, our performance on an 8K CM-2 is slightly less than one head of a Cray Y-MP.

Our work was hampered by inadequacies in the CM slice-wise Fortran compiler. There are additional opportunities for a coarser grained parallelism to be used on top of the fine grained parallelism, within the data parallel framework. Unfortunately, we could not exploit it because version 1.1 of the slice-wise compiler does not parallelize/vectorize the outer loop of a serial dimension. However, even without this additional parallelism we have demonstrated the viability of the fine grained approach using the current compiler technology. Future releases of the CM Fortran compiler may lead to further exploitation of parallelism, as will extensions of our work to the CM-5. We hope that future releases also include richer array section constructions, and more flexible alignment and layout directives; the lack of which led to considerable programming headache during the course of this work.

The paper is organized as follows. Section 2 gives a brief overview of the serial AMR

algorithm. Section 3 describes the data parallel version of the algorithm. The important points here are the data layout and grid generation. Section 4 discusses implementation details pertaining to the CM-2. Section 5 gives timings of the performance of the integrator alone and the overall AMR algorithm. We describe the numerical simulation of laser trenching in an integrated circuit substrate using a 2-D gas dynamics approximation of the physics in a simple geometry. In the conclusion we have several recommendations and a discussion of the parallelism we currently cannot exploit and its potential impact on the performance of the algorithm.

2 Overview of the AMR Algorithm

The AMR algorithm uses a nested sequence of rectangular grids to approximate the solution to a partial differential equation. The state variables are cell-centered, and an explicit finite volume scheme updates these values by computing fluxes at cell edges:

$$u_{i,j}^{n+1} = u_{i,j}^n - \Delta t \left[\frac{(F_{i+1/2,j} - F_{i-1/2,j})}{\Delta x} - \frac{(G_{i,j+1/2} - G_{i,j-1/2})}{\Delta y} \right]$$

When the solution resolution is insufficient, rather than refining a single grid cell at a time, rectangular fine grid patches are generated to cover those cells that need additional refinement. These grid patches have their own solution storage, with minimal storage overhead needed to describe the location and size of the grid itself. Grids are properly nested and aligned with each other, i.e. for every cell in a level l grid there is at least one level $l - 1$ cell surrounding it in all directions, although these coarse cells may belong to different grids. The grids are not rotated with respect to each other. Note that a fine grid can have more than one parent grid (see figure 1).

Each grid level has its own time step. Typically, if a grid is refined by a factor of 4, the time step is refined by a factor of 4 because of time step stability restrictions. The integration procedure on such a grid hierarchy then proceeds recursively: integrate on the coarse grid (ignoring fine grids); then use the coarse grid values with space time interpolation to provide boundary conditions for fine grids so that they, too, may be advanced in time. The algorithm is recursive in that the fine grid is in turn used in advancing still finer grids.

There are four separate components to the AMR algorithm that together generate this adaptive mesh hierarchy and advance the solution. For a complete description of the AMR algorithm see [1, 3]. The *error estimator* decides where the solution accuracy on a given grid level is insufficient and tags those grid cells as needing refinement. The *grid generator* creates mesh patches that cover all the flagged points. It takes as input the set of tagged points from the error estimator, and outputs a set of grid patches that together cover all the cells needing refinement. The inter-grid communication happens in the following two components. The *interpolation* routines initialize a solution on a new fine mesh, from

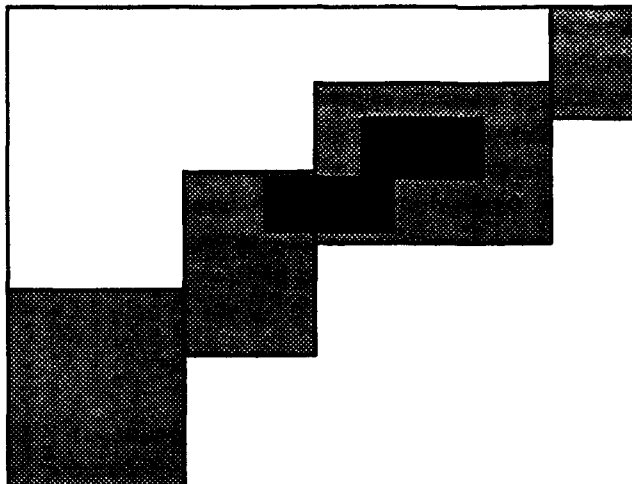


Figure 1: There is a single level 1 grid, level 2 has 4 grids and level 3 has 2 grids in this hierarchy.

either old fine meshes (injection) or coarser grids (interpolation), and they provide enough boundary values for fine meshes so that the integration stencil can be used at every interior point in a fine grid. The *flux correction* routine insures conservation at grid interfaces by modifying those coarse grid cells adjacent to a fine grid. We strictly enforce the condition that the flux out of a coarse grid cell during a single coarse time step equals the flux into the adjacent fine grid cells over all the corresponding fine time steps. This component also includes what we call *updating*; the fine grid cells update the coarse grid cell “underneath” using a conservative, volume weighted average of the fine grid values.

We typically use a high-order Godunov method to integrate a system of conservation laws [8, 9]. The integrator can be operator split or unsplit, the system of equations can be augmented by passively advected quantities, such as for multiple species, and the AMR shell doesn’t change.

3 The SIMD AMR Algorithm

In our data parallel version of the AMR algorithm, individual grid points rather than entire grids are mapped to processors. We make the restriction that all grid patches are a fixed size. This allows us to design a data layout scheme for mapping points to processors to minimize inter-grid global communication and preserve locality. This is the key idea in our approach.

The grid size restriction is motivated by the following considerations. When we first considered the mapping of grids to processors, we thought several grids of varying sizes

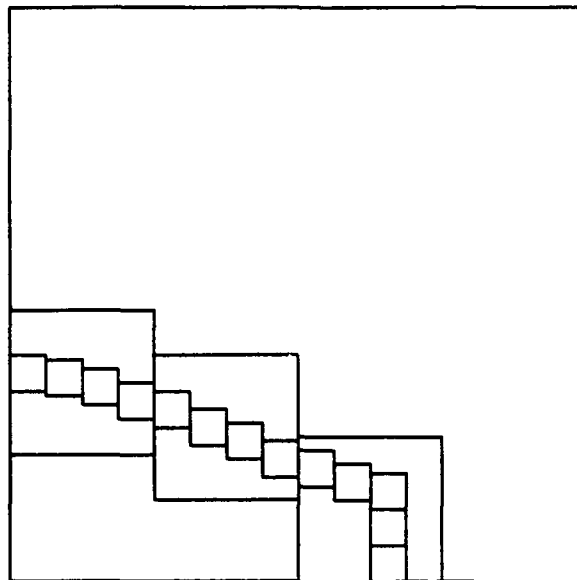


Figure 2: Three levels of grids are shown; each has the same number of grid points.

might be integrated at one time. A multi-dimensional type of bin packing could be used to group the fine grid patches to fill up a two-dimensional view of the machine. However, the CM-2 at NASA Ames Research Center has 32K processors. One quarter of the machine (the typical amount attached to a sequencer) has 8K processors, but only 256 floating point units. This corresponds to a 16 by 16 mesh. To keep the floating point pipes full requires a minimum of 4 grid points per processor. Hence a single 32 by 32 mesh can use one quarter of the machine and a single 64 by 64 grid patch can use the whole machine. With typical grid sizes in this range, a natural choice was to keep all grids the same size. In practice, on the order of 16-32 points per processor are usually needed for peak machine performance. Therefore, what fixed size is chosen is extremely important to the performance of the algorithm. Can we find a tile size small enough to use effectively in an adaptive setting, yet large enough to be integrated efficiently? In sections 4 and 5 we discuss the ramifications and efficiencies of this decision. Figure 2 shows an example of three levels of grids, each has the same number of grid points but smaller and smaller mesh widths, so they occupy a decreasing amount of physical space.

Given this restriction of fixed sizes grids (either 32 or 64 in each dimension), we have designed a data layout scheme for both the coarse and fine grids that preserves locality and minimizes communication. The layout is best described using the following indexing notation. We describe it using one space dimension. Higher dimensions use tensor products of the one-dimensional case.

Memory location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Fine Cell Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Coarse Cell Index	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16
Fine Cell Index	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Fine Cell Index	33	34	35	36	37	38	39	40	25	26	27	28	29	30	31	32

Table 1: Illustration of the mapping from grid points to memory location for the grids in figure 3.

Each coarse grid cell in the computational domain is given a unique index. If the computational domain consists of a single coarse grid with tile size 32, the cells are numbered from 1 to 32. If there are 2 coarse grids in the computational domain, adjacent to each other, the first is numbered from 1 to 32, and the second from 33 to 64. Suppose now there is a single coarse grid, and it is completely covered by fine grid patches. If the refinement ratio is 4, then 4 fine grids make up the computational domain. The i^{th} fine grid will be numbered from $(i-1)*32+1$ to $i*32$, corresponding to coarse grid cells $(i-1)*32/4+1$ to $(i-1)*32/4+8$. Of course, the fine grids do not have to start at location that are multiples of 32; for example a fine grid can be numbered from 25 to $25+32-1=56$.

Given this numbering convention, we can describe the two different layout strategies for the fine and coarse grids. (More precisely, the grid patch is mapped to a two-dimensional view of memory using the usual CM compiler default mapping, but we “interpret” it in a different way). Suppose for simplicity the tile size is 16. The finest grids are mapped to memory so that cell i is in memory location $i \bmod 16$, or more precisely, $(i-1) \bmod 16+1$. In other words, the grids are periodically wrapped as they are mapped to the CM memory so that adjacent cells from adjacent fine grids are in adjacent memory location. This keeps the injection operation from grids at the same level completely local.

The coarse grid layout is the complicated one. Suppose the refinement ratio is 4. Then cell 1 on the coarse grid corresponds to cells 1 through 4 on the fine level, coarse cell 2 to fine cells 5 through 8, etc. To keep the coarse/fine grid communication local, the rows and columns of the coarse grid are permuted so that no matter where the fine or coarse grid is located, corresponding cells are within 4 of each other. This enables fast data movement using nearest neighbor NEWS wires. For a single coarse grid as shown in figure 3, the grid point to memory mapping is shown in table 1. Note that even for the second fine grid, the corresponding coarse points are “nearby”. For example, point 17 on the fine grid corresponds to cell 5 on the coarse grid, and is one memory location away. This same correspondence holds regardless of where the fine grid is located, or how many coarse grids there are.

This coarse grid permutation is derived by sequentially distributing the coarse grid points, skipping 4 memory locations between points (for a refinement ratio of 4). When the

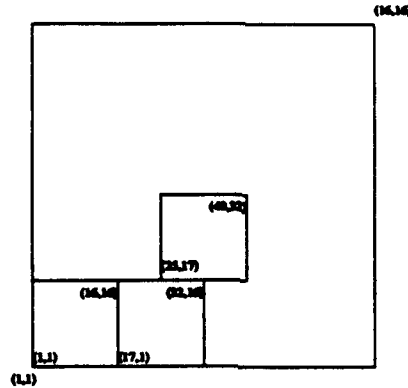


Figure 3: One coarse grid with three refined grids, corresponding to the memory layout in table 1. for a tile size of sixteen

end is reached, wrap around to the next available memory location from the beginning.

As often happens, an improvement to the serial AMR algorithm for proper nesting was discovered by rethinking the algorithm for the SIMD implementation. The proper nesting requirement for fine grids can be simply checked using a "domain calculus" with simple logical operations and a bitmap of the domain. The description of the set of all grids at a given level is simple to compute regardless of how complicated the domain at that level is. Given a domain at a certain level, properly nested subgrids must be one cell away from the boundary of the domain. Again this is simply computed with some shifts and logical operations. Previous (serial) implementations used complicated testing along fine grid boundaries to verify proper nesting.

The grid generation component of the AMR algorithm was greatly simplified by the use of fixed sized tiles. The serial version uses a pattern recognition algorithm to find "edges" in the tagged points; these edges form the edges of the new grid patches. (See [1] for details). Although rather sophisticated, this grid generation algorithm used negligible run time, and usually produced grids with approximately 80% efficiency, i.e., 80% of the points in the new grids were tagged, only 20% were additionally included to keep grids rectangular. The new grid generator simply tiles the smallest rectangular region bounding all the tagged cells using the fixed size tiles. It may happen that tiles have no tagged points underneath them, (for example, if the tagged points form a large circle and the tile is inside the circle). In this case the tile is deleted. This unsophisticated approach produces refined grids around the 50% efficiency level. Also, the use of fixed size tiles and the requirement of proper nesting may necessitate the use of overlapping grids. This rarely happens in practise, although it does slightly increase programming complexity.

In the original work of [3], Richardson extrapolation was used to estimate the error in the computation. This however had to be augmented by additional procedures depending

PatchSize	Two Dimensions		Three Dimensions	
	2 Ghost Cells	4 Ghost Cells	2 Ghost Cells	4 Ghost Cells
32	77%	56%	67%	42%
64	88%	77%	82%	67%

Table 2: Percentage of interior cells as a function of tile size

on the type of fluid flow being computed, for example, density gradients or compressibility and divergence were computed. We use only the latter, more ad hoc estimates in this work.

4 Implementation Details

The implementation decision with the most impact on performance concerned the treatment of ghost cells. To avoid special boundary stencils, each grid is surrounded by the additional number of points needed to apply the same finite volume stencil everywhere. Our simplified version of the high-order Godunov method uses 3 points to the side. These so-called ghost cells, or dummy cells, are obtained from adjacent fine grids or interpolated from coarser grids. Every grid contains space for these ghost cells along with the regular interior cells. For a grid of size 32, if there are 3 ghost cells on each side, only 26 cells are left for what we call the “real” interior grid cells. Since 26 is not divisible by 4 (our usual refinement ratio), in fact we must allocate 4 ghost cells on each side (all 4 are actually used in the more sophisticated version of the integrator).

Table 2 shows the fraction of interior cells as a function of tile size for 2 and 3 space dimensions. As can be seen, in 3 dimensional calculations for 32 sized tiles the fraction drops below 50%. In 2 space dimensions, 32 sized tiles is a possibility, although there are other reasons to prefer the larger sized tile.

In section 5, we measure the efficiency of the algorithm using grind time (the time to update one cell one timestep). The effective grind time is computed as the total CPU time divided by the number of interior zones. Strictly speaking, almost all the work for updating a cell must also be done at the first ghost cell, so this measure overestimates the grind time. This is due to the fact that for n cells, there are $n + 1$ fluxes, which use information from both cells adjacent to it. The other ghost cells however are only used to calculate slope information and to add a little extra artificial viscosity for problems with very strong shocks. We investigated several approaches to eliminating the permanent storage of these ghost cells, or alternatively, using slightly larger grids so that the interior grid mapped to the fixed size tile, with borders that wrapped around the processor array. However, given the restricted layout and alignment directives, these approaches were much slower than the 30 to 40% penalty paid by not using the ghost cell processors for the bulk of the computation.

This remains a problem with an unsatisfactory solution.

There are two ways we take advantage of the additional communication on the CM-2 beyond treating it as a mesh machine: we use Fastgraph and the power of 2 NEWS wires. Although the calculation is adaptive, with most of the communication patterns changing dynamically through the course of the computation, there is a pattern that does not depend on grid placement and can be precomputed. The coarse grid is stored in memory using the permuted, or shuffled memory layout described in section 3. However, every now and then the coarse grid itself is integrated. For this step the solution is unshuffled, and then reshuffled at the end of the integration step. Regardless of how many levels there are, or where the grids are located, the permutation is identical (modulo the periodic offset). We use the communication compiler Fastgraph [12] to precompute an optimized routing for the shuffling operation and its inverse. Fastgraph itself is expensive to use, but it is only called once, it depends only on the refinement ratio, and the routing may be saved between runs, so we do not include Fastgraph in our run time results. Fastgraph saves a factor of 2 in the shuffling time over using the router; this is approximately 5% of the total run time on a typical run. Other users have reported much larger time savings when using the communication compiler; we surmise that because the permutation itself is so simple, the router does a good job of it to begin with. Nevertheless, for other classes of architectures besides a CM-2, and other applications besides AMR, an optimized preconfigurable routing would be extremely useful.

The second way we use the CM-2 as more than a mesh machine is with shifts (CSHIFT, EOSHIFT) of more than 1 location at a time. This is useful in the updating step of the AMR algorithm. For example, to compute the average of a 4 by 4 sub-block of fine grid cells so that the result may be injected onto the coarse grid, we sum by using a shift of one and then two in each direction. In the reverse procedure, the sum is logarithmically spread back to all 16 fine cells, so that the one lying on top of the coarse cell can do the actual injection. This reduces the number of shifts from 3 to 2 in each dimension for the sum and the spread, with the run time for this routine reduced by 1/3 as well. Even the NEWS network communication is much more expensive than floating point operations, and the updating routines involve very little computation. We also experimented with segmented scans (adds and copies), but found our implementation faster for such small segment lengths as 2 or 4.

5 Numerical Results

We present two types of measurement to indicate the performance of the AMR algorithm on the CM-2. First, we demonstrate the performance of the integrator as a function of grid size, without any adaptivity. The performance of the overall algorithm can be no better than this, since we count all other CPU time as overhead in the algorithm. Next we show

PatchSize	1 by 1		2 by 2		4 by 4	
	Integ. time	BC time	Integ. time	BC time	Integ. time	BC time
32	1.93	.96	7.76	2.27	30.69	3.07
64	5.40	3.25	21.55	7.56	86.45	19.34
128	18.60	7.34	74.55	29.05	298.82	73.41
256	70.63	20.87	281.56	73.84	1125.93	289.27

Table 3: CPU times with no refinement as a function of grid size and number of grids

numerical results and timings for experiments with 3 levels of grids. All the timings use the CM timer with version 1.1 of the slicewise compiler with optimization, on one sequencer attached to 8K processors of the CM-2 at NASA Ames Research Center. Our results show only the CM CPU time; the elapsed time (the sum of CPU and idle time) varies greatly, depending on how heavily loaded the CM/front end is. Our best idle times are typically between 2 and 5% of CPU time, depending slightly on the grid size.

Table 3 shows the CPU time for the integrator and the periodic boundary condition routine as a function of tile size. For this experiment we do not use any refinement (i.e. only coarse grids at the base level), but we do use several grids at that level. Notice that for patch sizes of 64 or more, doubling the number of points in each dimension gives integration times that increase by a factor of $18.60/5.40 = 3.4$, and $70.63/18.60 = 3.7$, unfortunately less than, though close to 4. When we go from a patch size of 32 to 64, this is far from the case ($5.40/1.93 = 2.80$). Although we would prefer to use the smaller patch size for greater efficiency in resolving localized flow features in the solution, the extra cost of integration on such a small patch makes this choice infeasible, at least for the current compilers. For 3 dimensional calculations, we expect this to change.

The time for the periodic boundary conditions is sublinear. With one coarse grid, periodic boundary conditions must be applied at all four sides of the grid. This type of boundary condition is the most expensive since data must be shifted approximately halfway across the computational domain due to the ghost cells. (Reflecting wall boundary conditions are less expensive, but since they must be computed inside the integrator, the interpretation of the integration run-time as a function of tile size becomes more complicated.) When there are several coarse grids in the computational domain, each grid also gets (interior) boundary conditions from the adjacent grids, which is an efficient operation.

The grind times for this problem (microseconds per cell per update) are summarized in table 4. For a given tile size, the grind times decrease due to the decreasing cost of the periodic boundary conditions. For a fixed number of base grids, (looking down a column), the grind times decrease due to the better performance of the CM-2 on larger blocks of data. We feel that reasonable performance is possible with 64 sized patches, and use that

PatchSize	1 by 1	2 by 2	4 by 4
32	52	45	38
64	28	24	22
128	19	19	17
256	15	15	15

Table 4: μ secs to update one zone for one timestep (includes all overhead) as a function of tile size

size in our the two dimensional calculations with adaptive refinement. Although the grind time for a 64 sized patch is 50% more than the grind time for a 256 sized patch, the latter has 16 times the number of points. The total cost of the computation will be cheaper using smaller sized patches as long as there are fewer than 8 of them. (We hope to improve this break even point with future release of the compiler).

We wrap up this section by describing a calculation modeling the flow of a hot dense gas leaving a square trench into a low density and temperature medium. This calculation is a prototype for modeling laser deposition of energy into an integrated circuit substrate. Ultimately lasers will be used to dig micron scale trenches in integrated circuits. Before this can be done, it is important to understand the dynamics of the laser induced flow so that debris patterns can be categorized or even predicted as a function of energy deposition.

The setup of this problem is straightforward. The entire computational region is embedded in a unit square. Within this region, a box of size $1/8$ wide by $7/8$ height is cut out of the computational domain from the upper left corner. This rectangle is a void region, meaning it is not part of the computational region, but is part of the solid wall exterior boundary. All boundaries of the computational domain are set to be reflecting boundaries, so no fluid escapes from the computational region. The $1/8$ by $1/8$ region in the lower left has initial density of 1.0 and pressure 10.0. The $7/8$ wide by 1 high region in the right part of the domain, called the ambient region, has initial density and pressure set to 0.1 and 1.0 respectively. The velocities everywhere in the domain are initially set to zero and an ideal equation of state with a γ of 1.4 is used to relate density, internal energy and pressure. An illustration of the problem setup is shown in figure 4.

The major feature of the flow at late time (illustrated in figure 5) is a large bow shock penetrating into the ambient fluid. Even at this early time, the initial conditions look like a point source for the bow shock as the shock is already quite spherical. Behind the shock is the contact discontinuity delineating the boundary between the hot gas and the ambient gas. As the hot gas flows out into the ambient region, a shear layer forms causing the vortical rollup of the gas. The size of the rollup region will play an important part in the size of the debris regions around the trench.

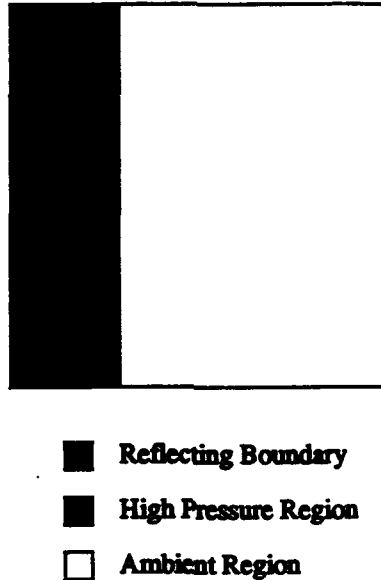
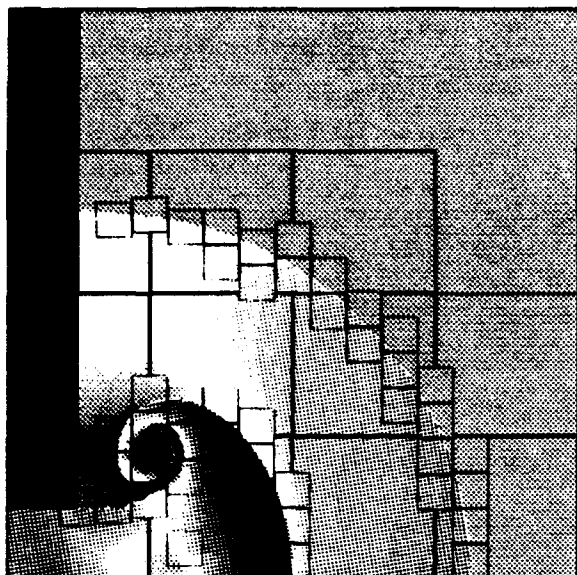


Figure 4: A schematic of the initial conditions for the prototype calculation modeling laser deposition of energy into an integrated circuit substrate.

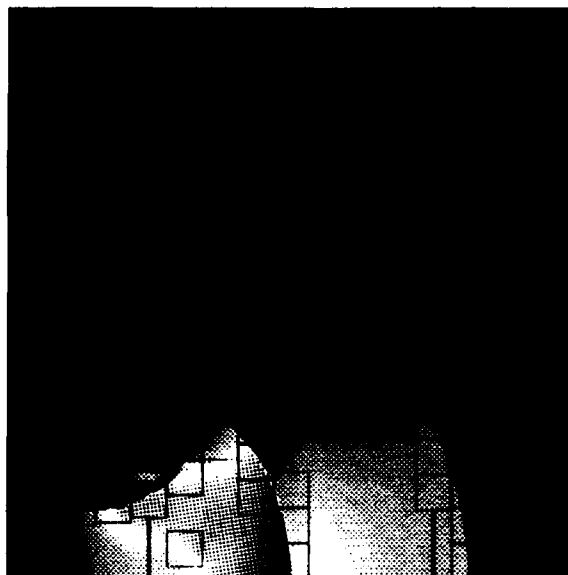
Table 5 is a breakdown of CPU times for the AMR code when run for ten coarse grid time steps. The bulk of the time taken outside of the integration routine is taken by the boundary conditions routines. Within this category, 72 % of the time is taken interpolating from coarse grids to fine grids. The grind time for this calculation is 25.1 microseconds/cell which fits within the bounds shown in table 4. Although the grind time is a little under a factor of two for the best uniform case (256 x 256), the savings in computation costs is still great as only a fraction of the computation region is computed using a fine mesh. The number of cells advanced by the AMR algorithm is only 12% of the number of cells that would need to be advanced by the uniform case. Therefore a factor of four overall gain in efficiency is achieved.

6 Conclusions

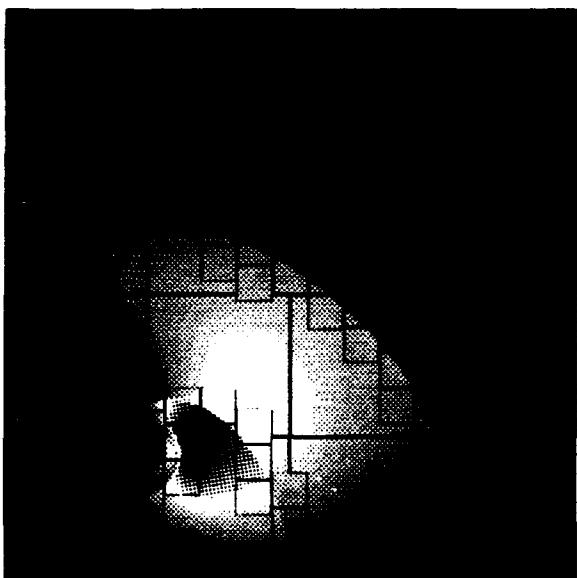
Despite the disappointing performance of the CM-2 for small grid sizes, we feel that the two dimensional implementation of AMR is a useful tool. Our overall performance on an 8K CM-2 is roughly equivalent to a single head of the Y-MP. We hope in the future that richer array constructs, and layout and alignment directives will make possible further improvements in the efficiency. Although our current choice of tile size does not scale to use the full machine in an efficient manner, we believe that three dimensional calculations,



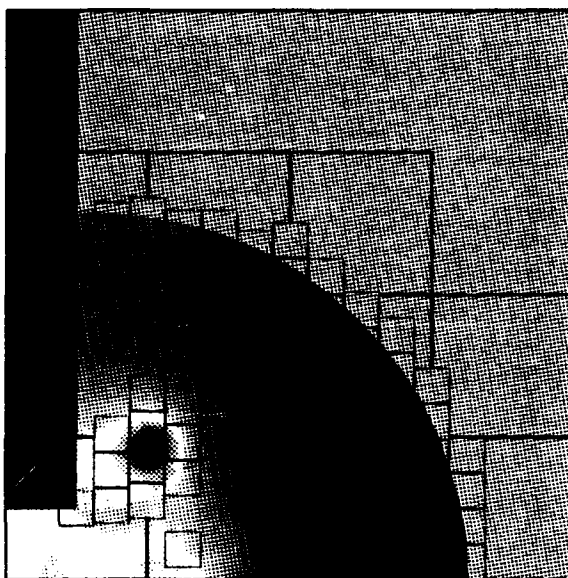
(a)



(b)



(c)



(d)

Figure 5: Prototype integrated circuit problem results at time .124 and cycle 70. Figure (a) is a raster plot of the logarithm of density, (b) and (c) are plots of x and y momentum respectively, and (d) is a plot of the logarithm of total energy. Values range from the low color blue (cold) to the high color red (hot).

Routine	CPU Time	Percent
Integration	587.79	76%
Boundary Cond.	107.35	14%
Cell Updates	10.92	1%
Error Est./Regrid	49.01	6%
Flux Correction	40.96	5%
Fastgraph	23.03	-

Table 5: Distribution of CPU usage by category within the AMR code for the integrated circuit calculation.

(or improved two dimensional calculations with tile size 32) would scale to use the full Connection Machine. Research along these directions is in progress.

Our greatest disappointment in working on the CM-2 came from its inability to exploit a coarser grained parallelism we found, on top of the fine grained data parallelism that is the basis of our approach. The source of this additional parallelism is the multiple grids at any given level. For example, instead of integrating one grid at a time, integrate all grids at a given level. Exactly the same operations are done to integrate any grid. We were particularly excited about the possibility of parallelizing the boundary condition routines. Ghost cells might possibly cause a load imbalance, but the periodic offsets in the layout of the grids would mitigate this by distributing the locations of the ghost cells in memory. When grids are shifted, the ghost cells often end up interior to the grid, rather than around the perimeter of the grid. Figure 6 illustrates this schematically.

Unfortunately, version 1.1 of the compiler does not parallelize (vectorize) across serial dimensions, and all our attempts to restructure the do loops to force it to do so failed. By integrating (interpolating, updating, etc.) many patches at a single time, we feel the 32 sized tile would be practical, allowing greater overall efficiency in the AMR algorithm. In fact, when a fully operational version of the CM-5 becomes available this coarser grained parallelism would make an interesting case study in a hybrid SIMD/MIMD model of computation.

Acknowledgements. It is a pleasure to acknowledge discussions with Scott Baden, Yanmin Sun, and Steve Hammond during the course of this work. M. Berger was supported in part by DOE grant DE-FG02-88ER25053, by AFOSR grant 91-0063, and by a PYI, NSF ASC-8858101. Additional support was provided in part by Cooperative Agreement NCC2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA). Work was performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center. J. Saltzman was

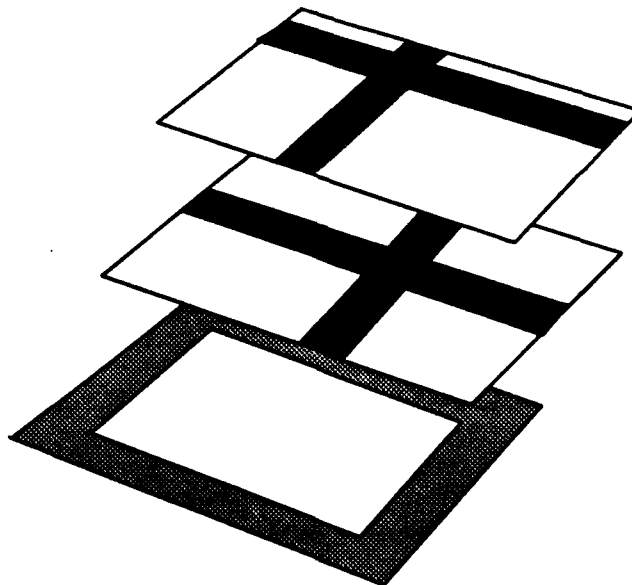


Figure 6: The boundary ghost cells are shaded on each grid patch. They are distributed to a different memory location in each patch.

supported by the U.S. Department of Energy through Los Alamos National Laboratory under contract No. W-7405-ENG-36 with additional support from the Center for Research in Parallel Computation and DARPA/NSF grant DMS-8919074 UC-Berkeley. We would like to thank the NAS systems division at NASA Ames for access to their Connection Machine CM-2.

References

- [1] J. Bell, M. J. Berger, J. Saltzman, and M. Welcome. Three dimensional adaptive mesh refinement for hyperbolic conservation laws. Preprint UCRL-JC-108794, Lawrence Livermore National Laboratory, Livermore, CA, December 1991.
- [2] M. J. Berger. Adaptive mesh refinement for parallel processors. In *Proceedings of 1987 SIAM Conference on Parallel Processing*, Los Angeles, CA, December 1987.
- [3] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64-84, 1989.
- [4] M. J. Berger and J. Olinger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:482-512, 1984.

- [5] R. Biswas. *Parallel and Adaptive Methods for Hyperbolic PDES*. PhD thesis, RPI, 1991.
- [6] R. Biswas and J. Flaherty. Parallel and adaptive methods for hyperbolic partial differential equations. In *Proceedings of the 7th IMACS International Conference on Computer Methods for Partial Differential Equations*, New Brunswick, NJ, June 1992.
- [7] J. Cerutti and H. Trease. The free-Lagrange method on the Connection Machine. In W. C. H. Trease, J. Fritts, editor, *Proceedings of the Next Free-Lagrange Conference*, Copper Mountain, Colorado, 1991.
- [8] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *Journal of Computational Physics*, 87:171-200, 1990.
- [9] P. Colella and P. Woodward. The piecewise parabolic method (ppm) for gas-dynamical simulations. *Journal of Computational Physics*, 54(1):174-201, April 1984.
- [10] W. Crutchfield. Load balancing irregular algorithms. Preprint UCRL-JC-107679, Lawrence Livermore National Laboratory, Livermore, CA, July 1991.
- [11] W. Crutchfield. Parallel adaptive mesh refinement: An example of parallel data encapsulation. Preprint UCRL-JC-107680, Lawrence Livermore National Laboratory, Livermore, CA, July 1991. Submitted to *Concurrency, Practice and Experience*.
- [12] E. D. Dahl. Mapping and compiled communication on the Connection Machine system. In *Proceedings of The Fifth Distributed Memory Computing Conference*, April 1990.
- [13] W. Gropp and D. Keyes. Semi-structured refinement and parallel domain decomposition methods. In J. S. P. Mehrotra and R. Voigt, editors, *Unstructured Scientific Computation on Scalable Multiprocessors*. MIT Press, 1992.
- [14] S. W. Hammond and T. J. Barth. An efficient massively parallel Euler solver for unstructured grids. *AIAA Journal*, 30(4):947-952, April 1992.
- [15] L. F. Henderson, P. Colella, and E. G. Puckett. On the refraction of shock waves at a slow-fast gas interface. *Journal of Fluid Mechanics*, 224:1-27, 1991.
- [16] W. D. Hillis. *The Connection Machine*. MIT Press, 1985.
- [17] Y. Kallinderis and A. Vidwans. Parallel adaptive-grid navier-stokes algorithm development and implementation. Technical report, University of Texas at Austin, 1992. Preprint.

- [18] O. Lubeck, M. Simmons, and H. Wasserman. The performance realities of massively parallel processors: A case study. Preprint LA-UR-92-1463, Los Alamos National Laboratory, Los Alamos, NM, 1992. Submitted to IEEE Supercomputing '92.
- [19] S. McCormick and D. Quinlan. Dynamic grid refinement for partial differential equations on parallel computers. In *Proceedings of 7th International Conference on Finite Element Methods in Flow Problems*. University of Alabama, April 1989.
- [20] J. Quirk. *An adaptive grid algorithm for computational shock hydrodynamics*. PhD thesis, College of Aeronautics, Cranfield Institute of Technology, 1991.
- [21] M. Ruffert. Collisions between a white dwarf and a main-sequence star II. Simulations using multiple nested refined grids. Preprint MPA 657, Max Planck Institute for Astrophysics, Garching, March 1992. Submitted to Astronomy and Astrophysics.
- [22] R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3(5):457-481, October 1991.